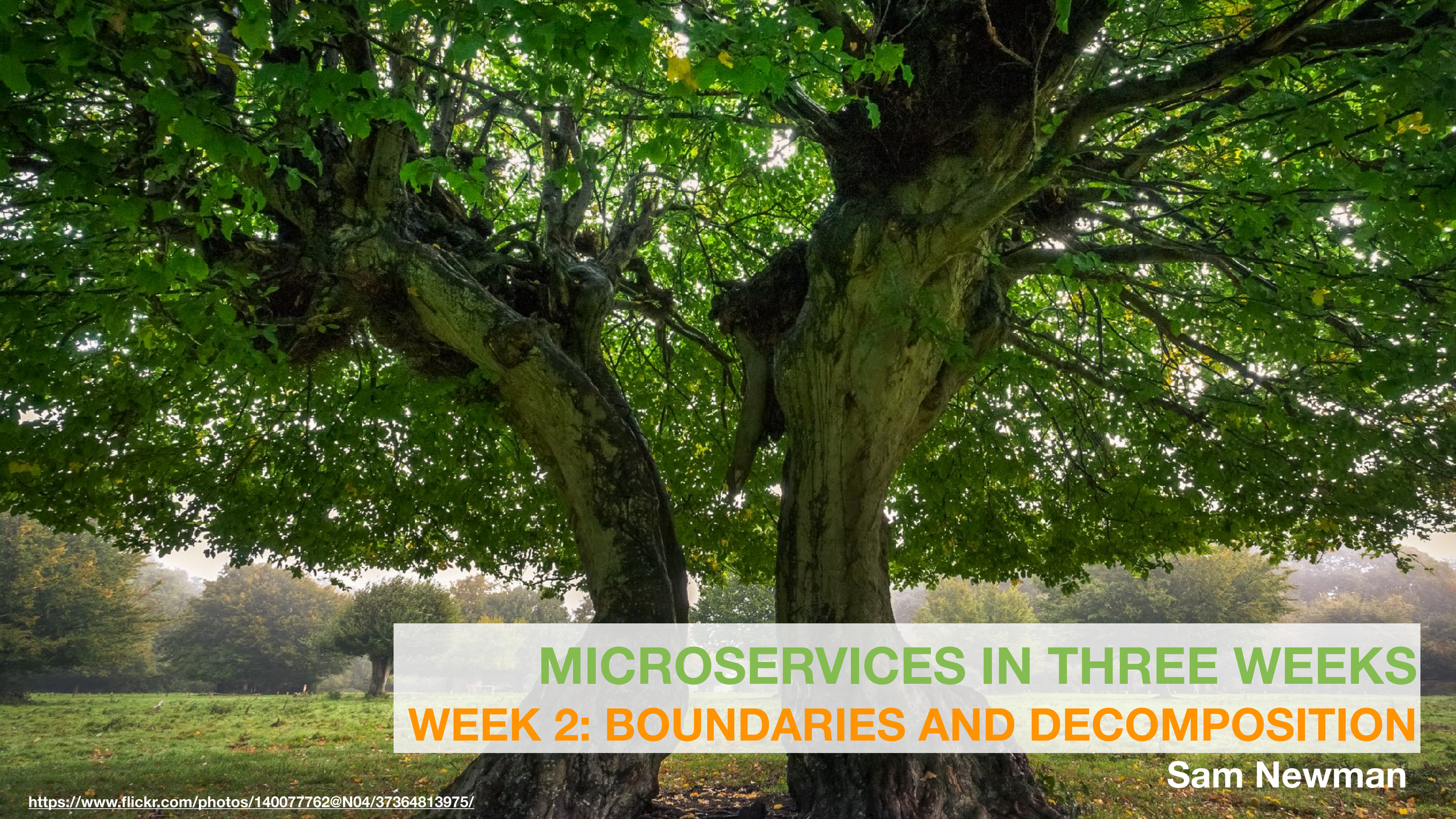


# MICROSERVICES IN THREE WEEKS
# WEEK 2: BOUNDARIES AND DECOMPOSITION

Sam Newman

https://www.flickr.com/photos/140077762@N04/37364813975/

# This is week two of three!

# Each session is three hours long

Each session is three hours long

We will have two 10-15min breaks

# Please ask questions using the Q&A widget

# Ownership and Org Structure

## WHAT WE'LL COVER

# Ownership and Org Structure

# Domain-Driven Design (in brief)

## WHAT WE'LL COVER

Ownership and Org Structure

Domain-Driven Design (in brief)

Planning a migration

## WHAT WE'LL COVER

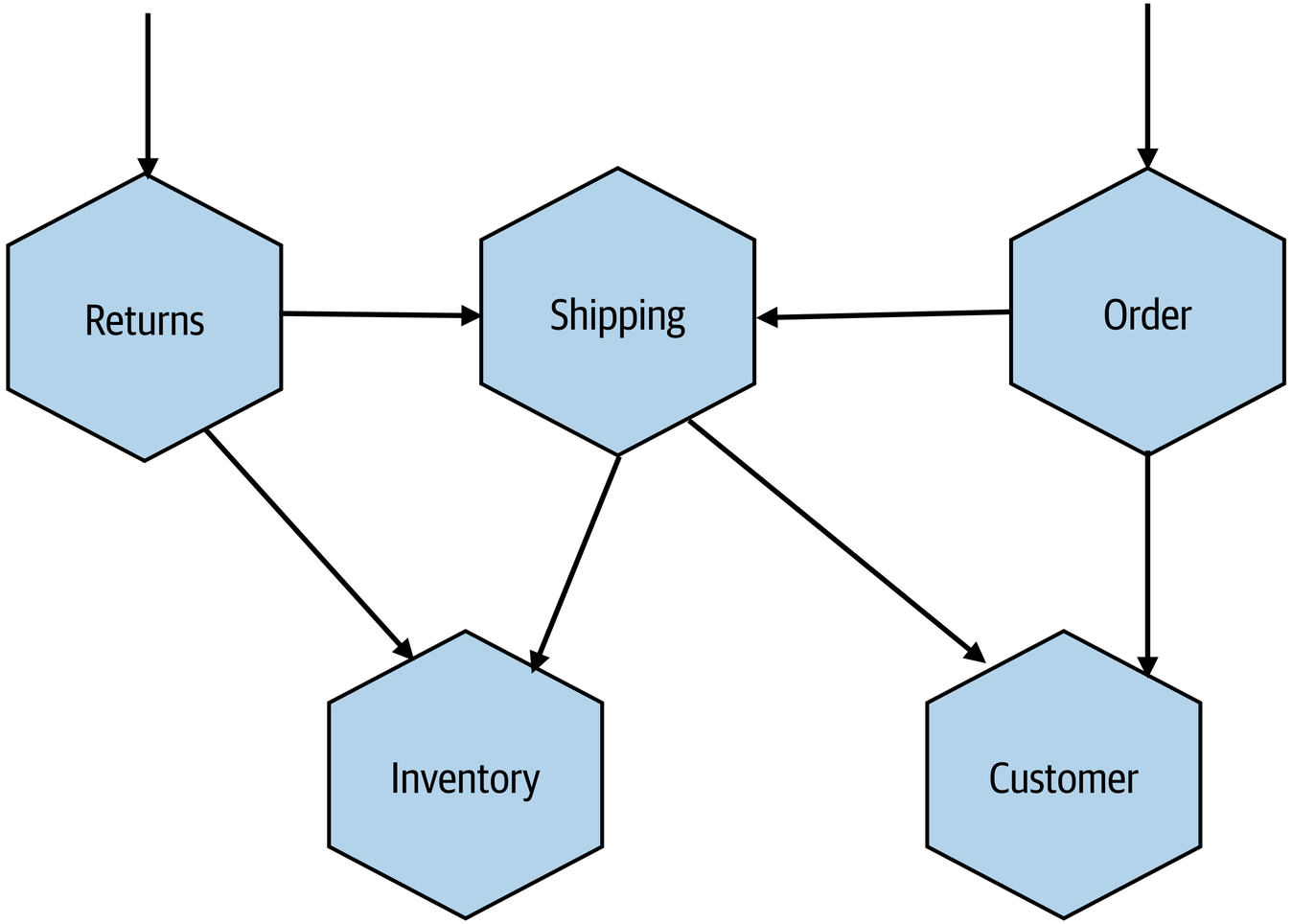
Ownership and Org Structure

Domain-Driven Design (in brief)

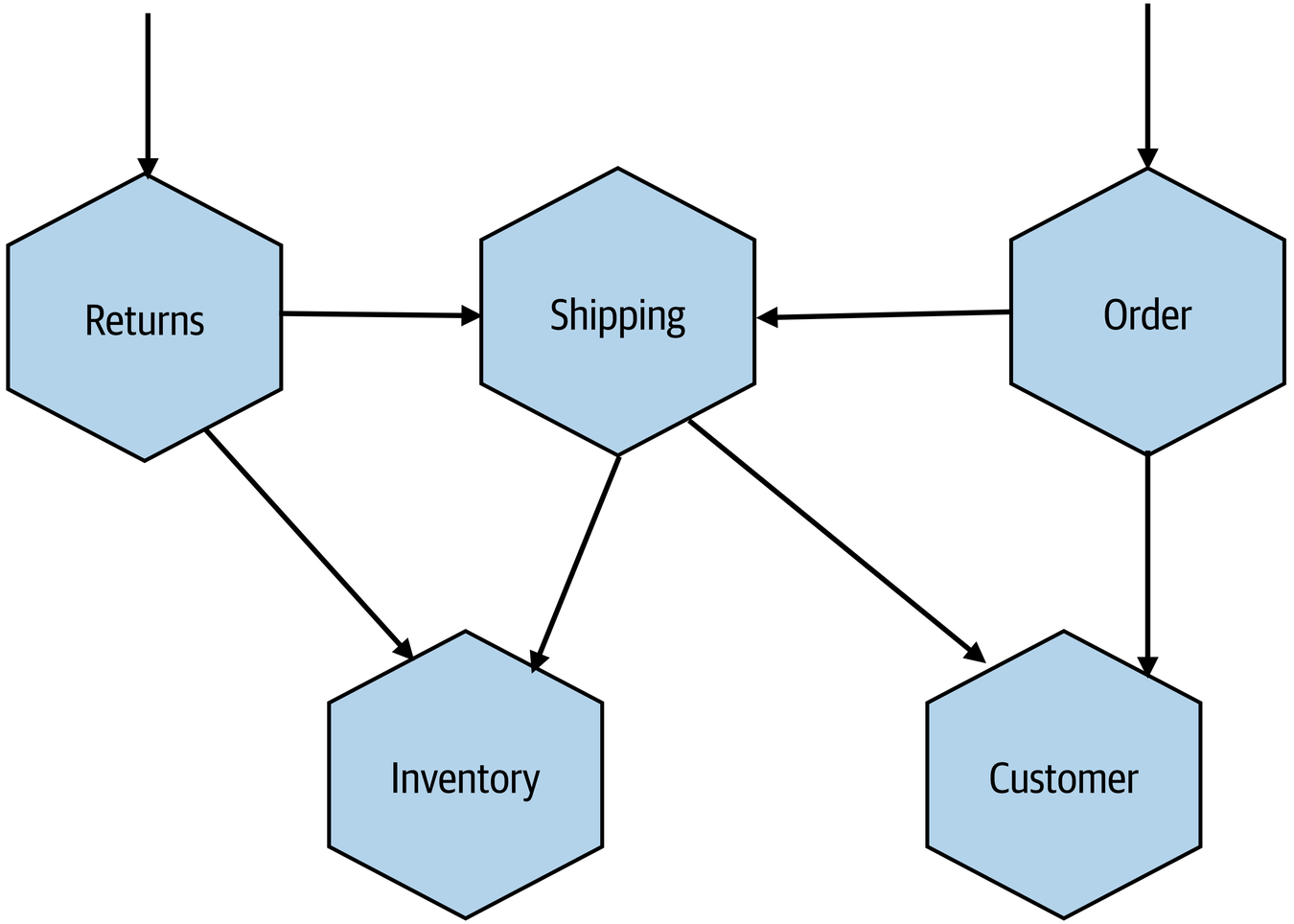
Planning a migration

Application Decomposition Patterns

# BRIEF RECAP…

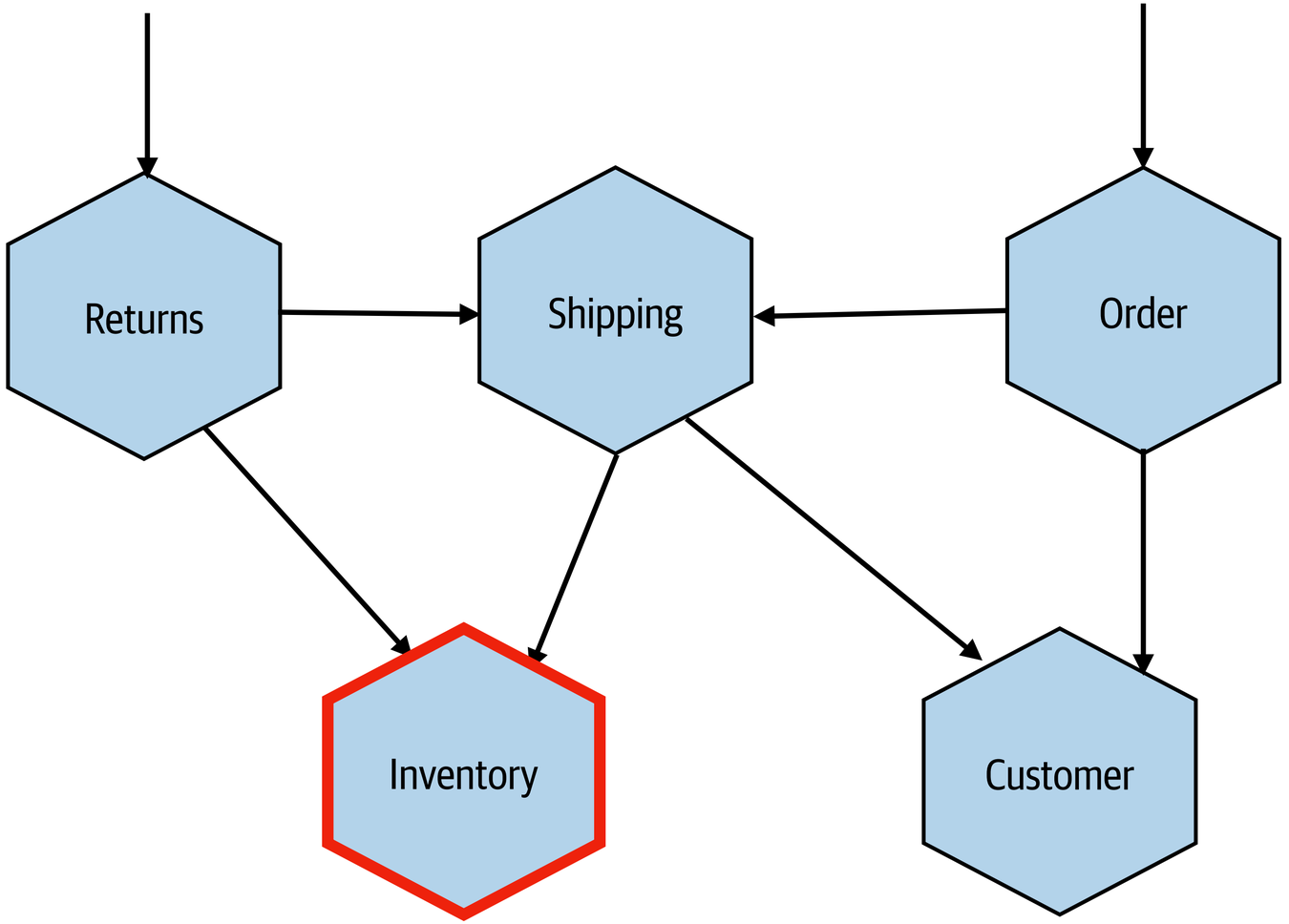


## BRIEF RECAP…



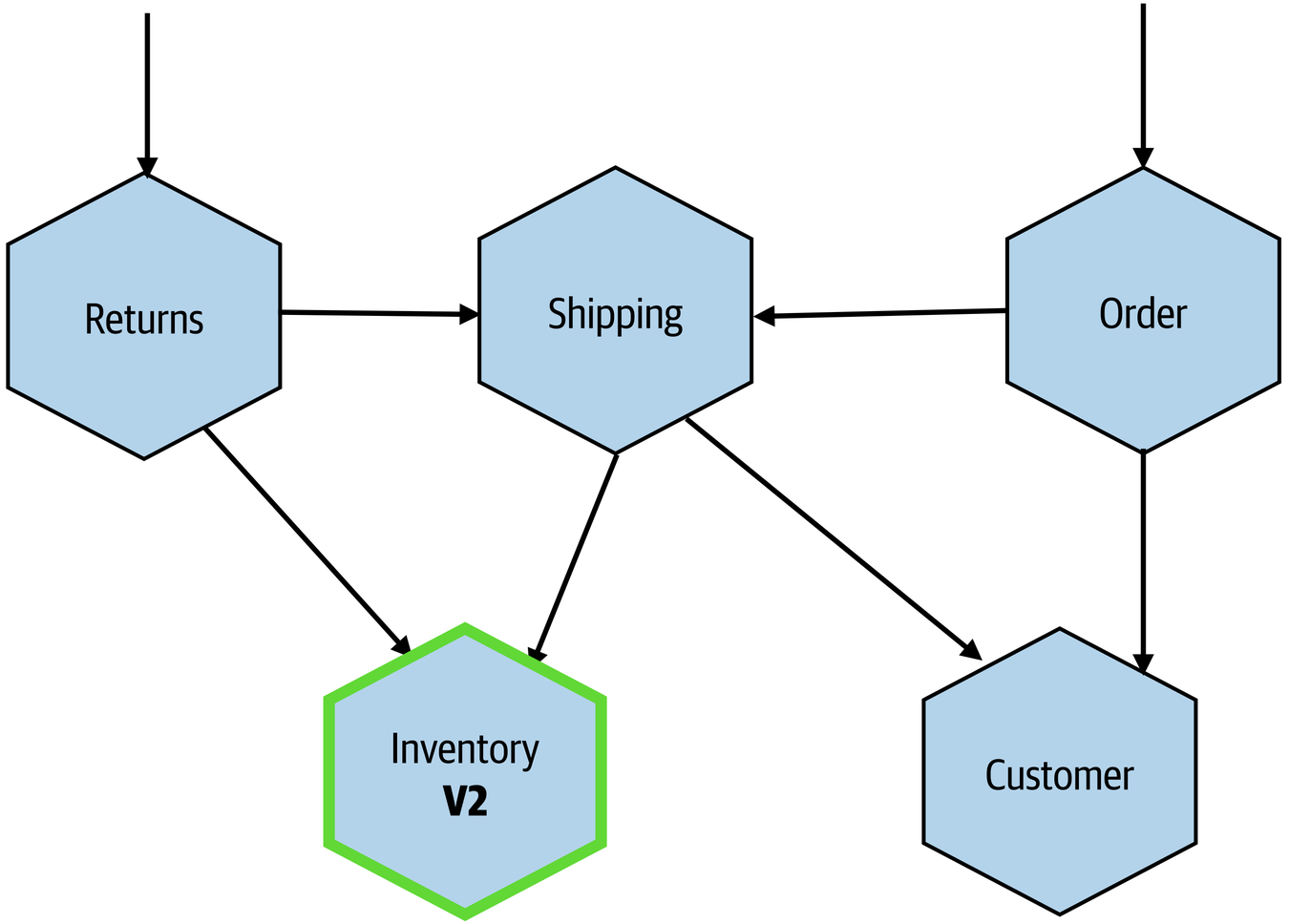

**Independently Deployable**

## BRIEF RECAP…



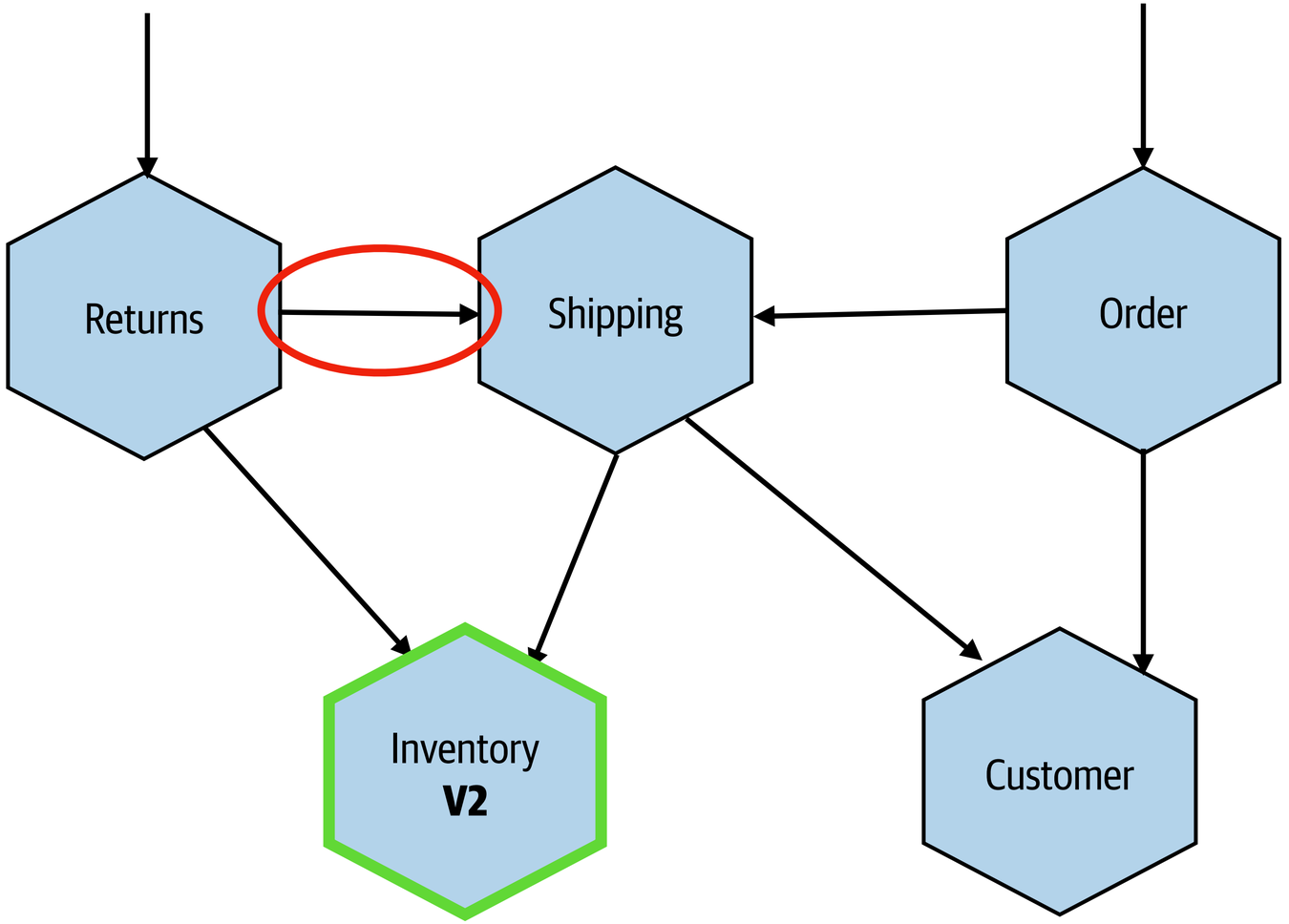

**Independently Deployable**

# BRIEF RECAP…




**Independently Deployable**

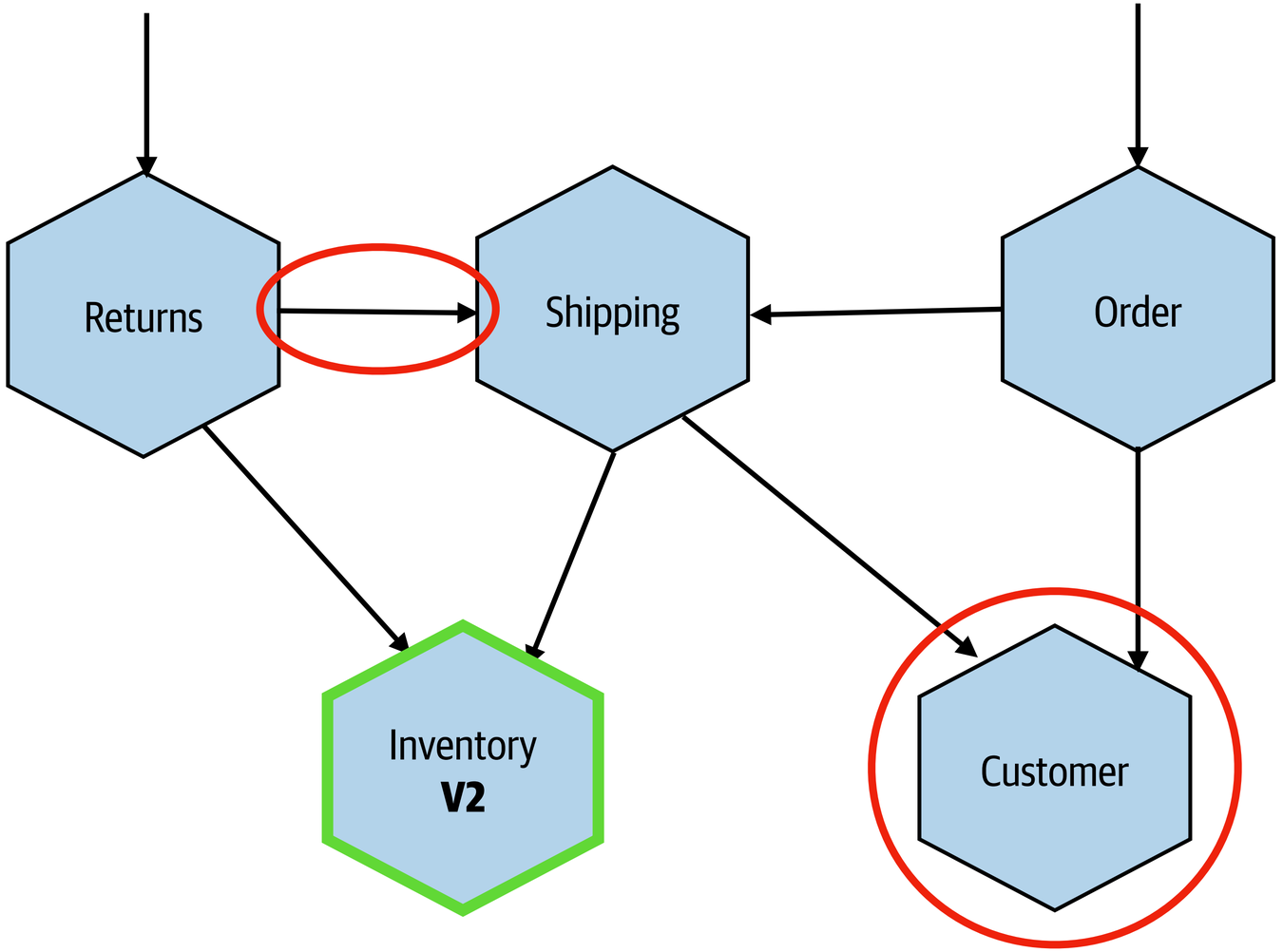
# BRIEF RECAP…




Independently Deployable

Microservices often depend on other microservices

# BRIEF RECAP…




Independently Deployable

Microservices often depend on other microservices

Boundaries primarily defined by the domain

# Questions from last week?

## WHAT WE'LL COVER

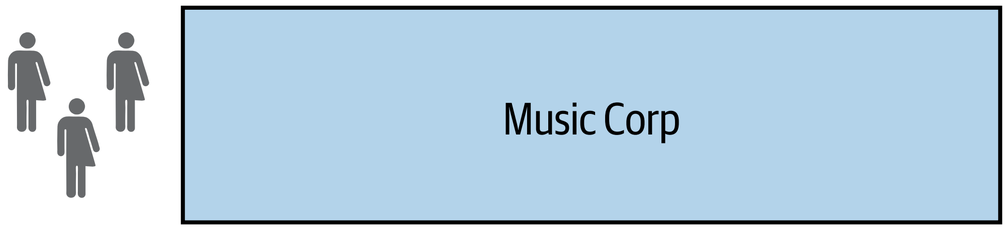
Ownership and Org Structure

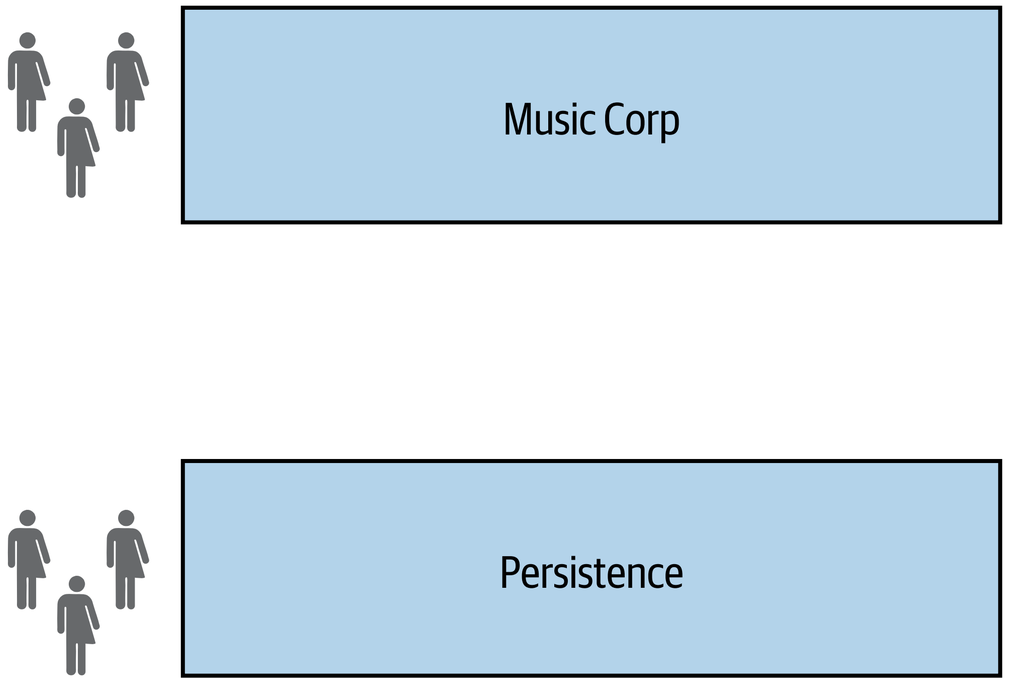
Domain-Driven Design (in brief)

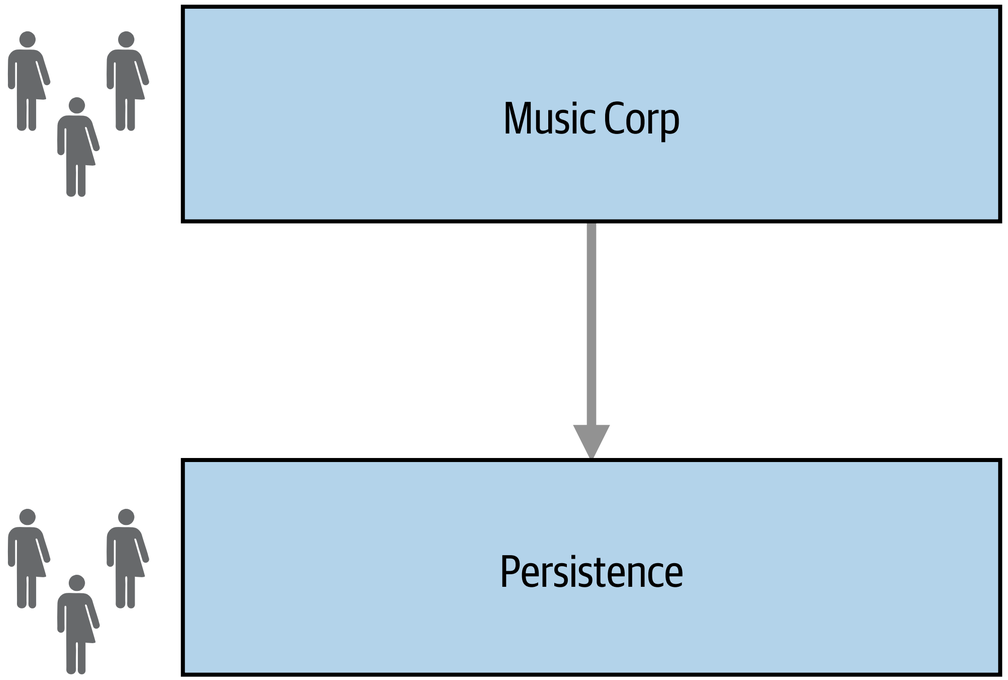
Planning a migration

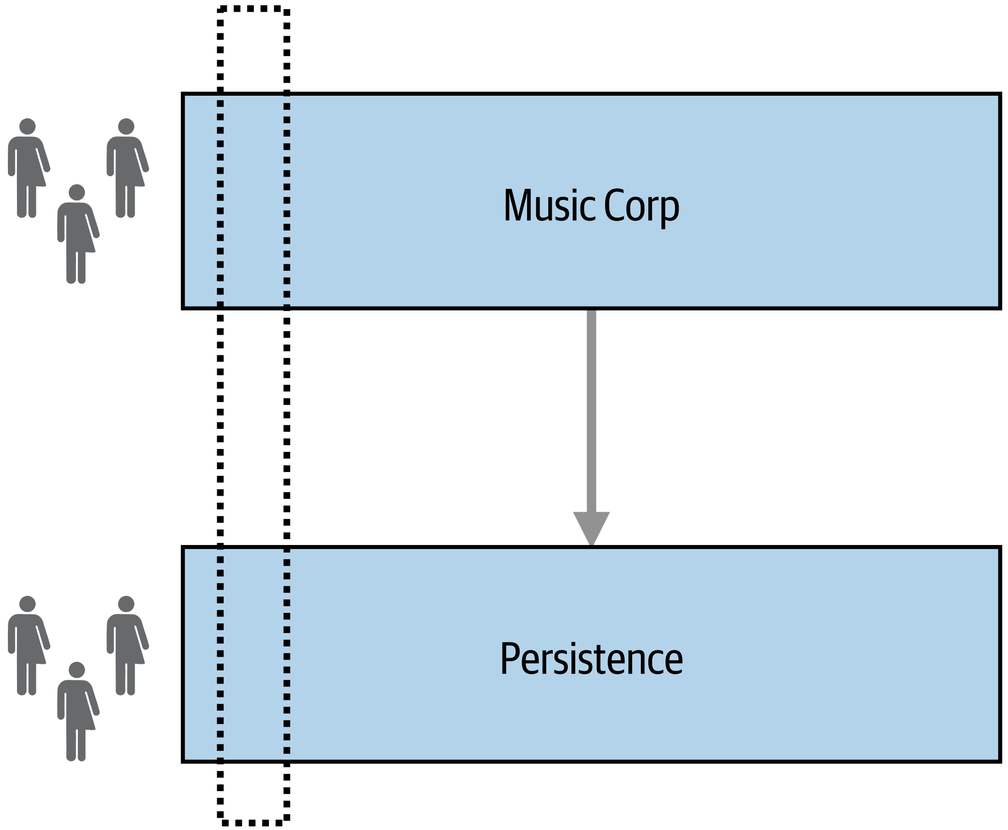
Application Decomposition Patterns

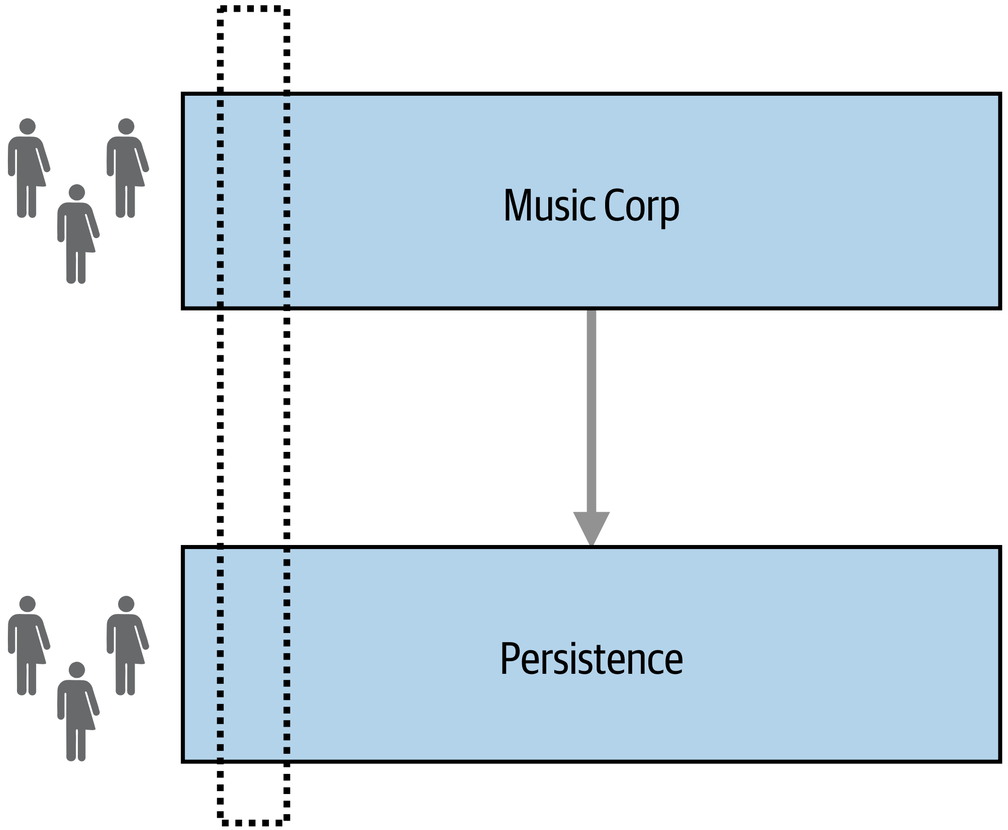
# 1/4 Ownership and Org Structure


Music Corp

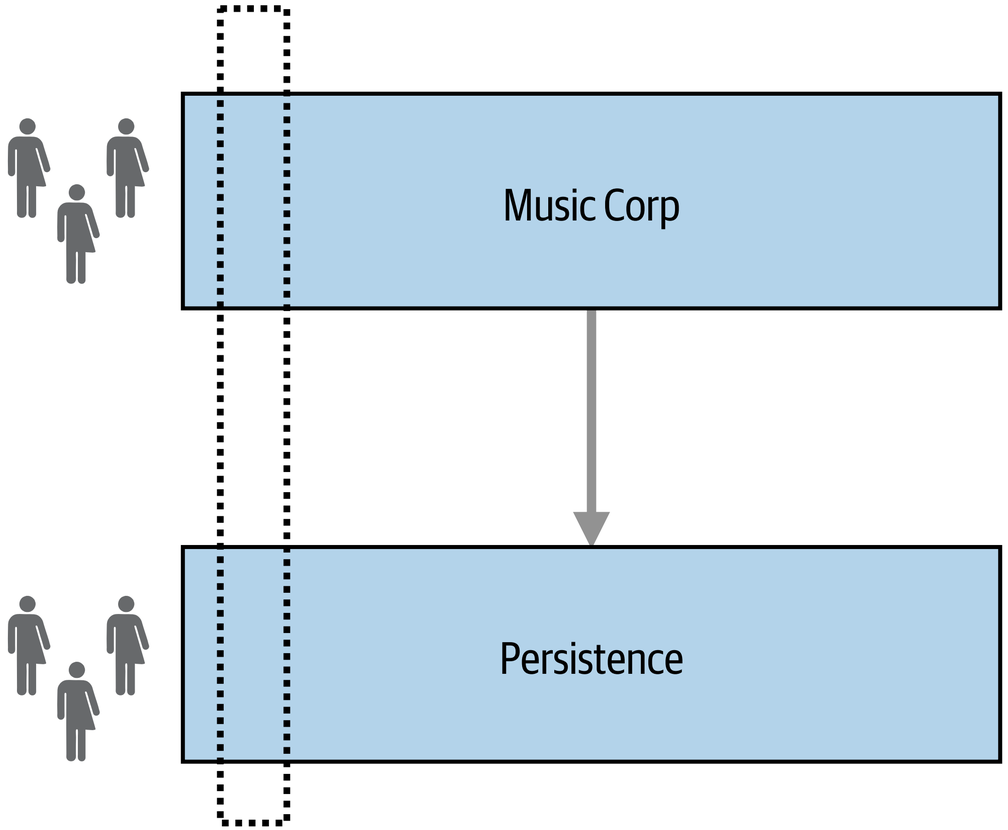

Music Corp
Persistence


Music Corp
Persistence

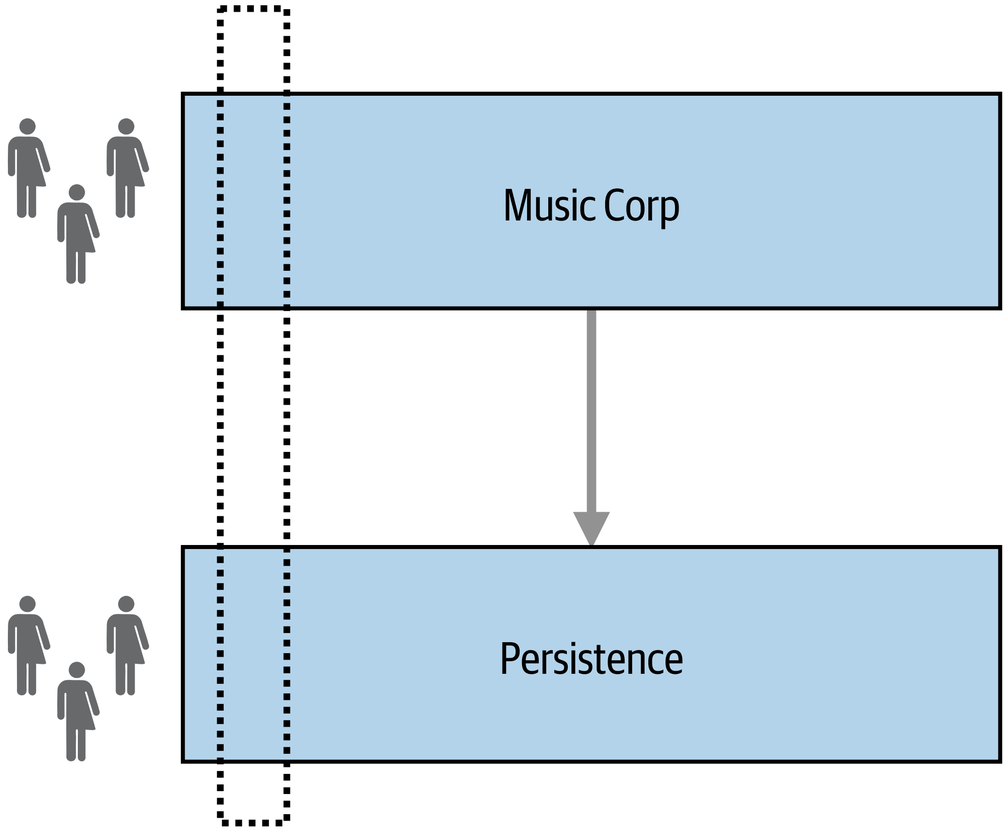

Music Corp
Persistence


Music Corp
Persistence

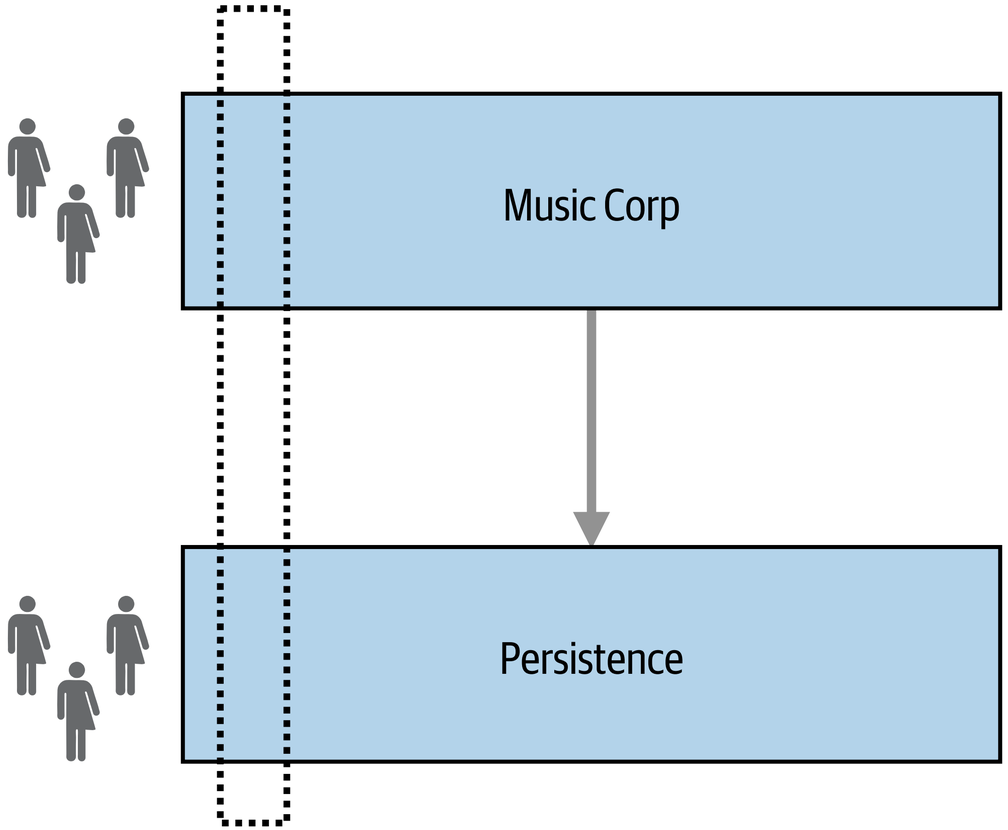

Genre:
OK

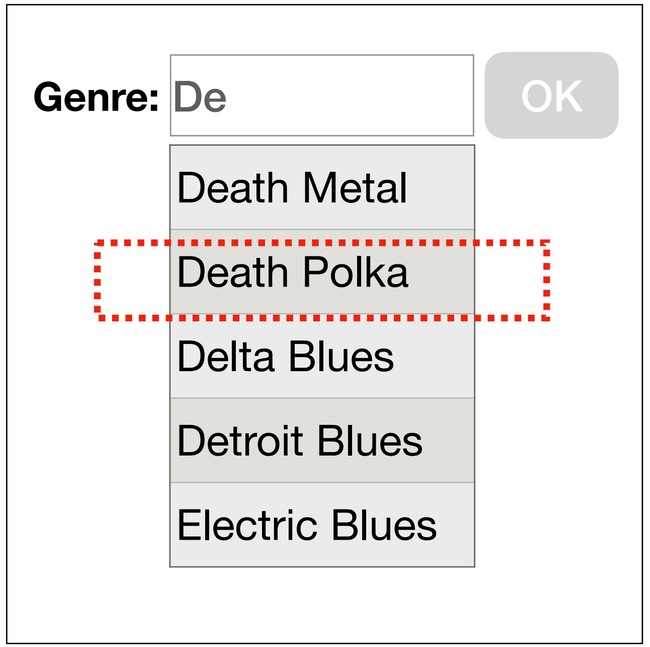

Music Corp
Persistence

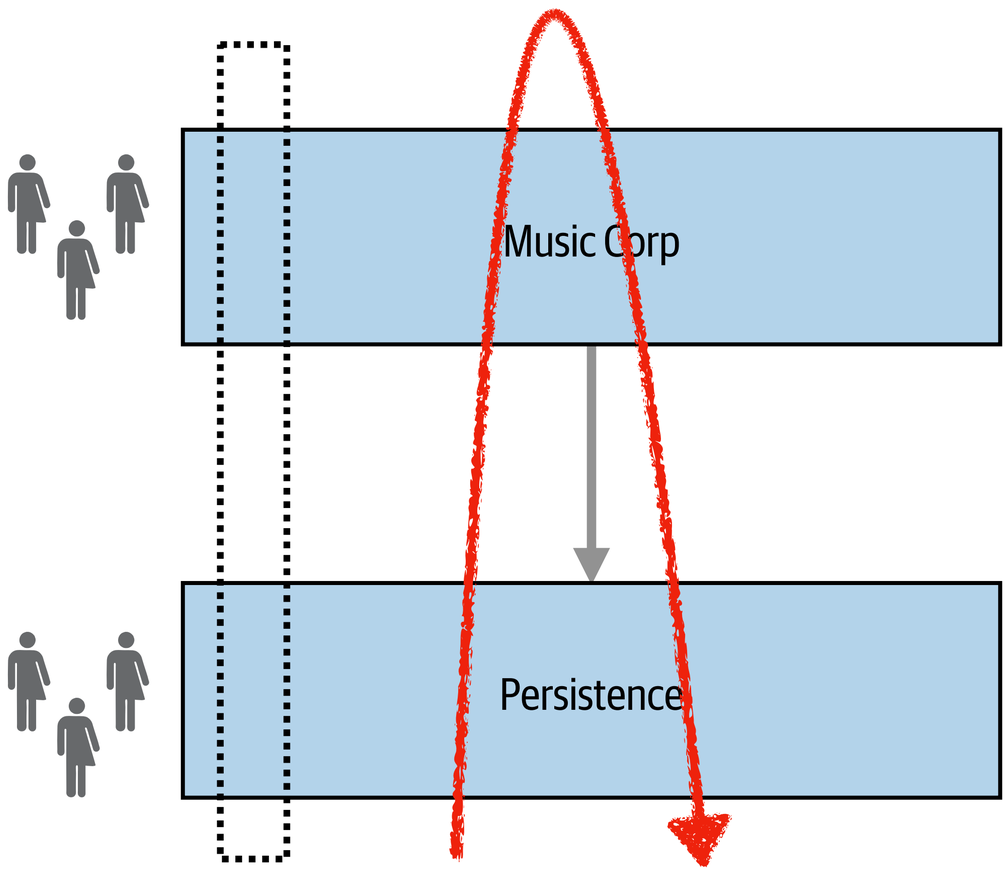

Genre:
De
OK

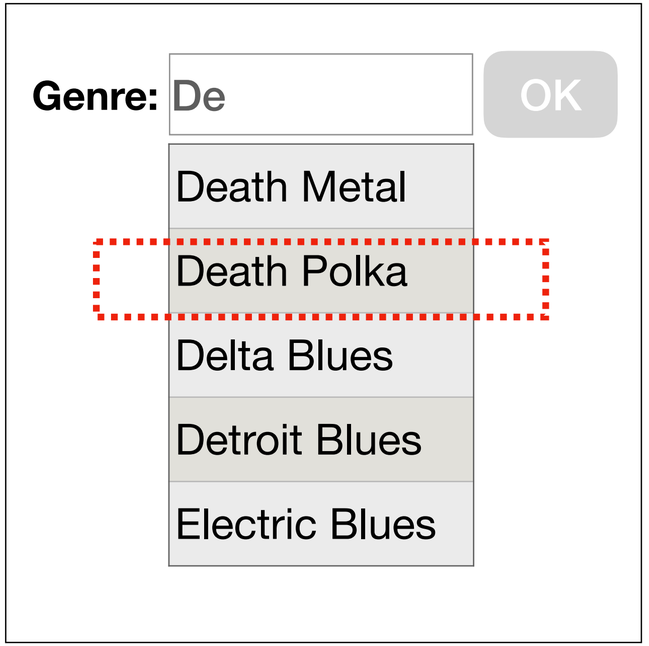

Music Corp
Persistence

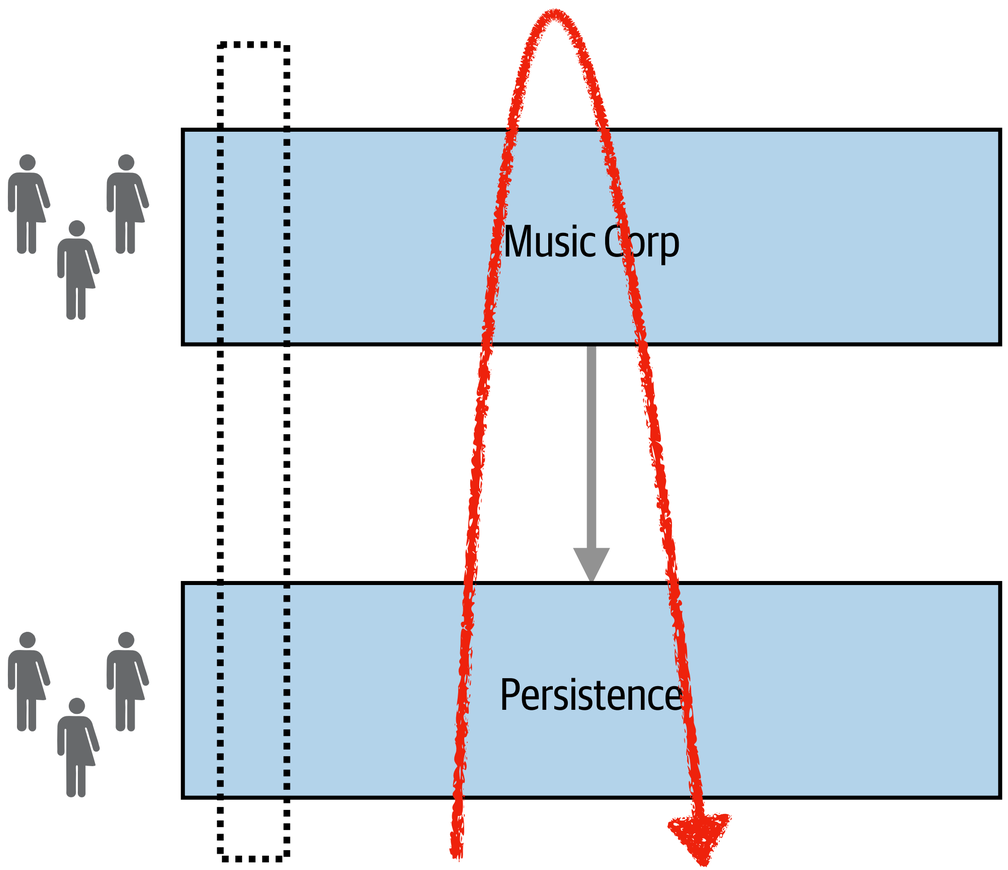

Genre:
De
OK
Death Metal
Death Polka
Delta Blues
Detroit Blues
Electric Blues


Music Corp
Persistence

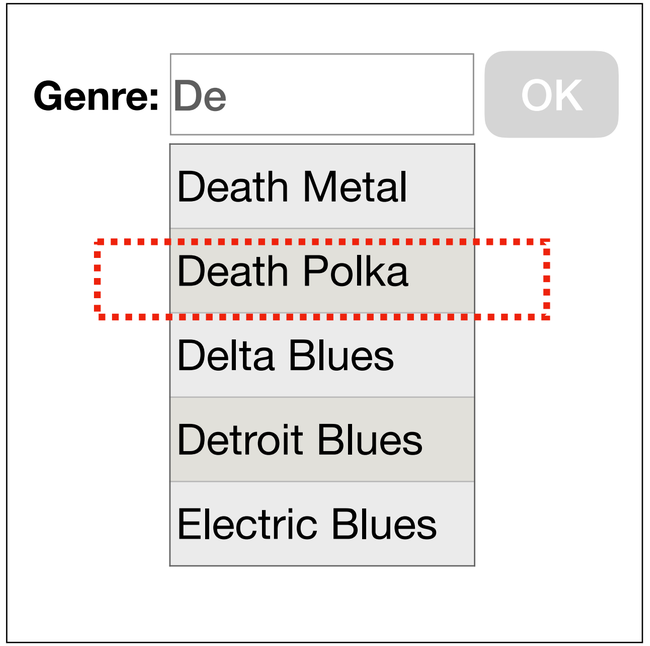

Genre:
De
OK
Death Metal
Death Polka
Delta Blues
Detroit Blues
Electric Blues

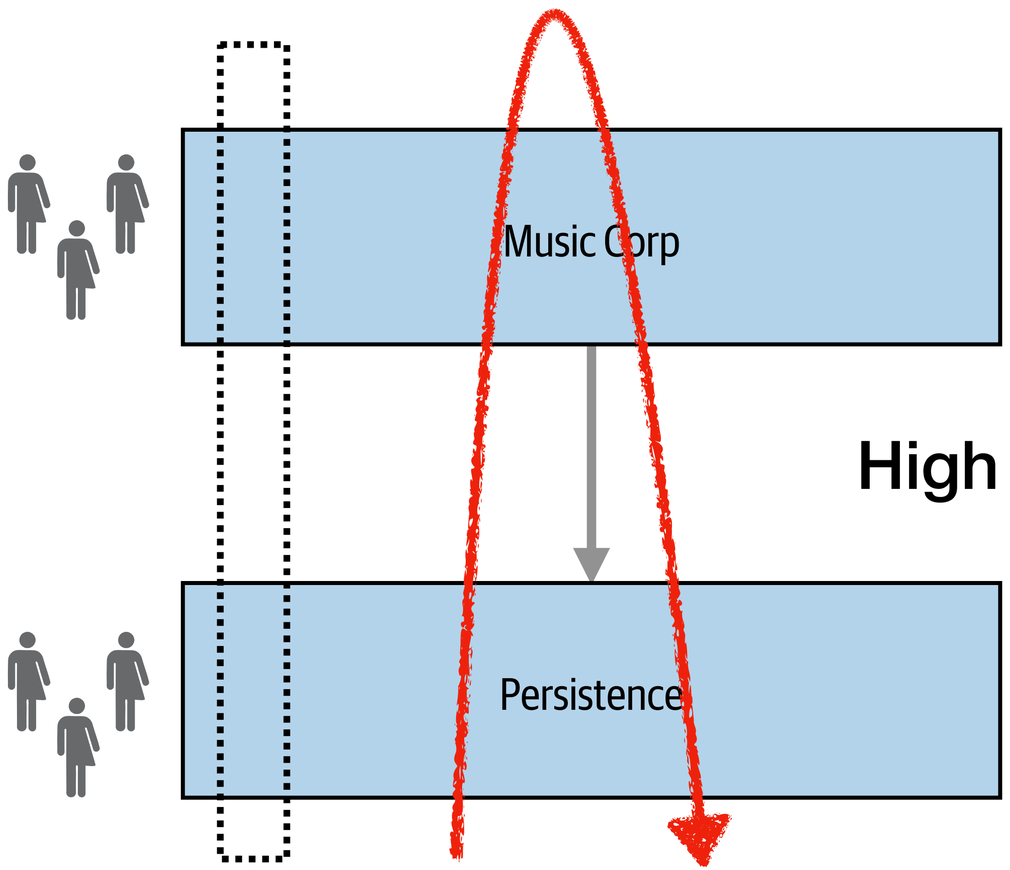

Music Corp
Persistence


Genre:
De
OK
Death Metal
Death Polka
Delta Blues
Detroit Blues
Electric Blues



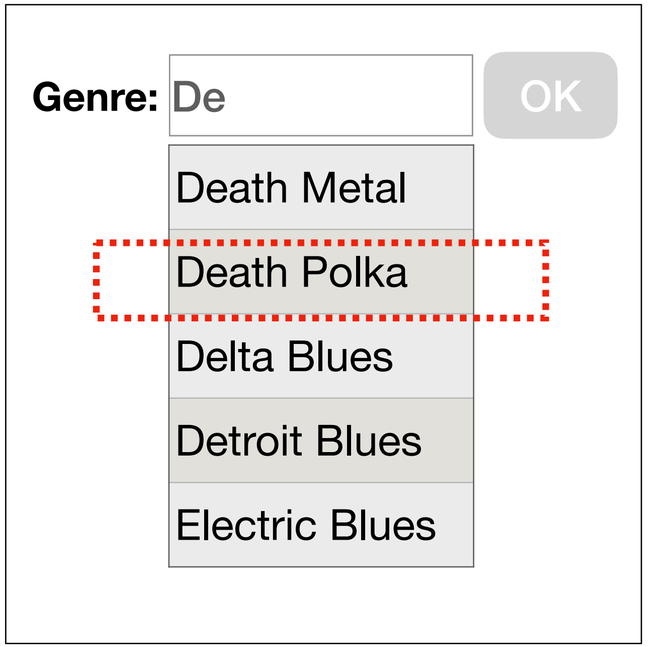

Low Cohesion

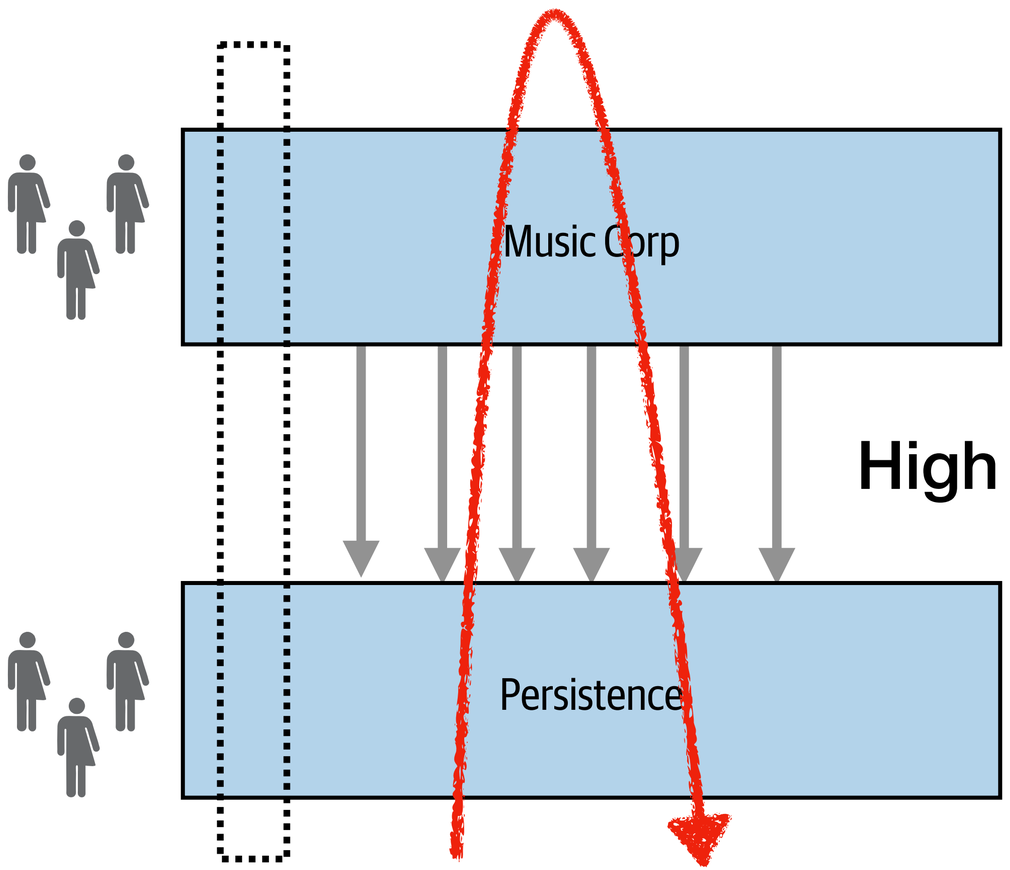





High Coupling

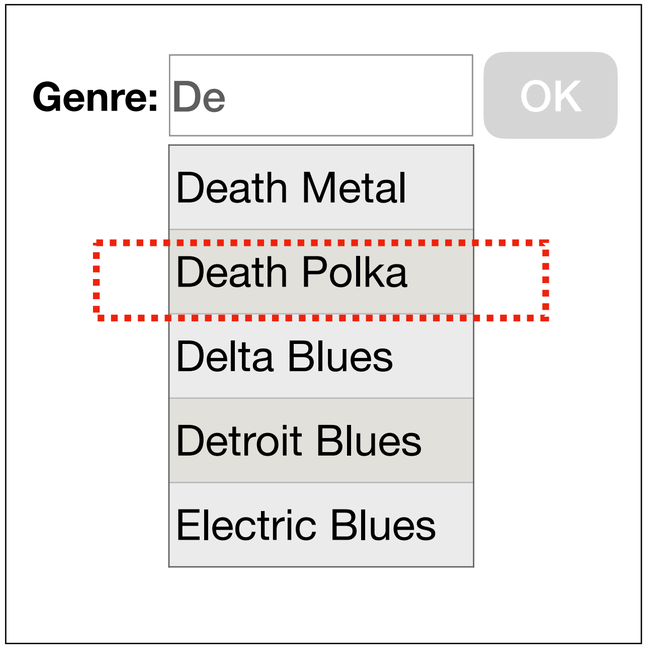
Low Cohesion





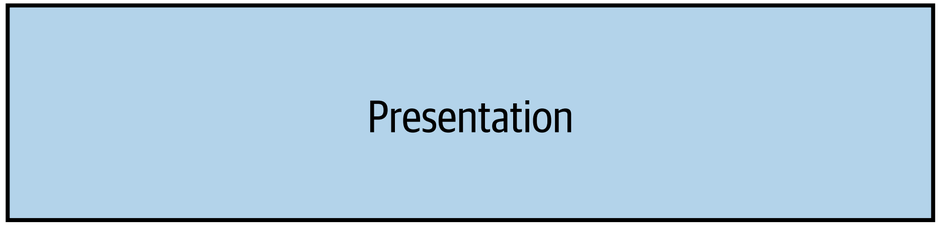

High Coupling

Low Cohesion

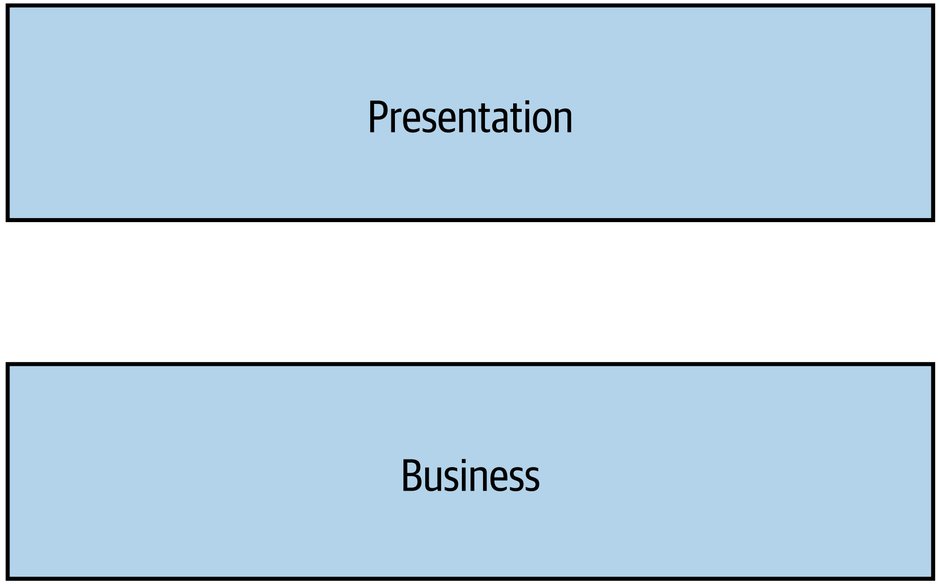


# THREE TIERED ARCHITECTURE

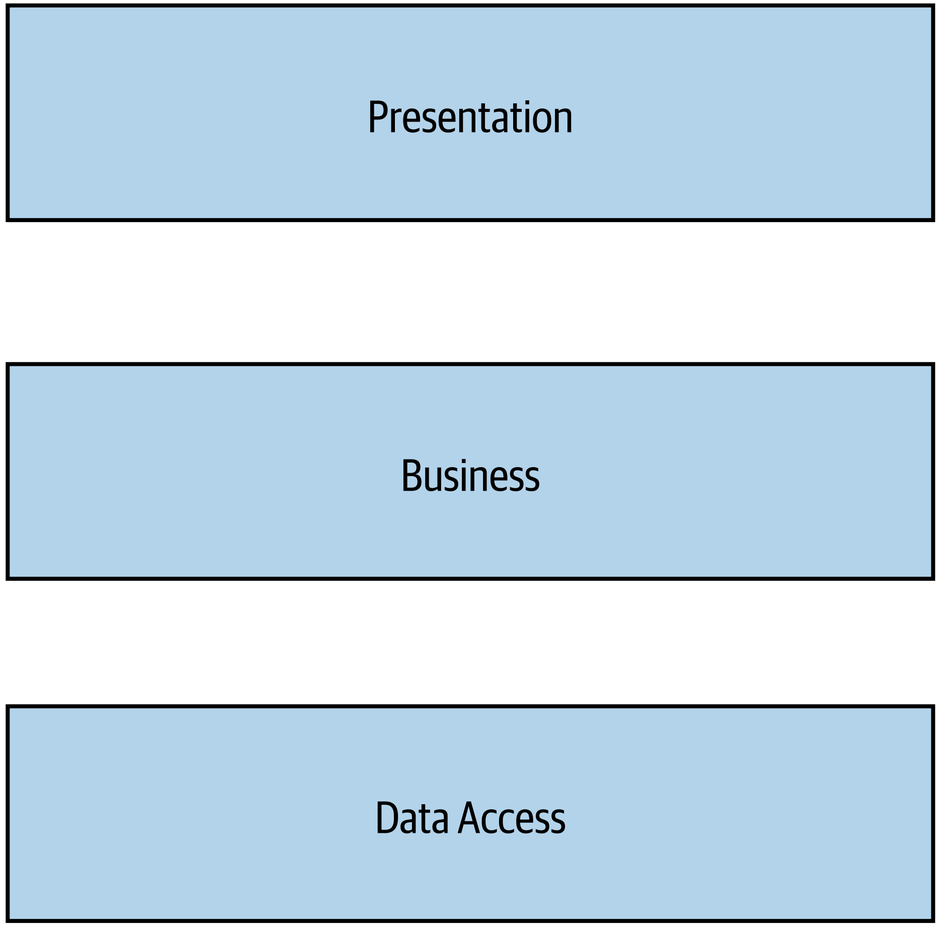
# THREE TIERED ARCHITECTURE



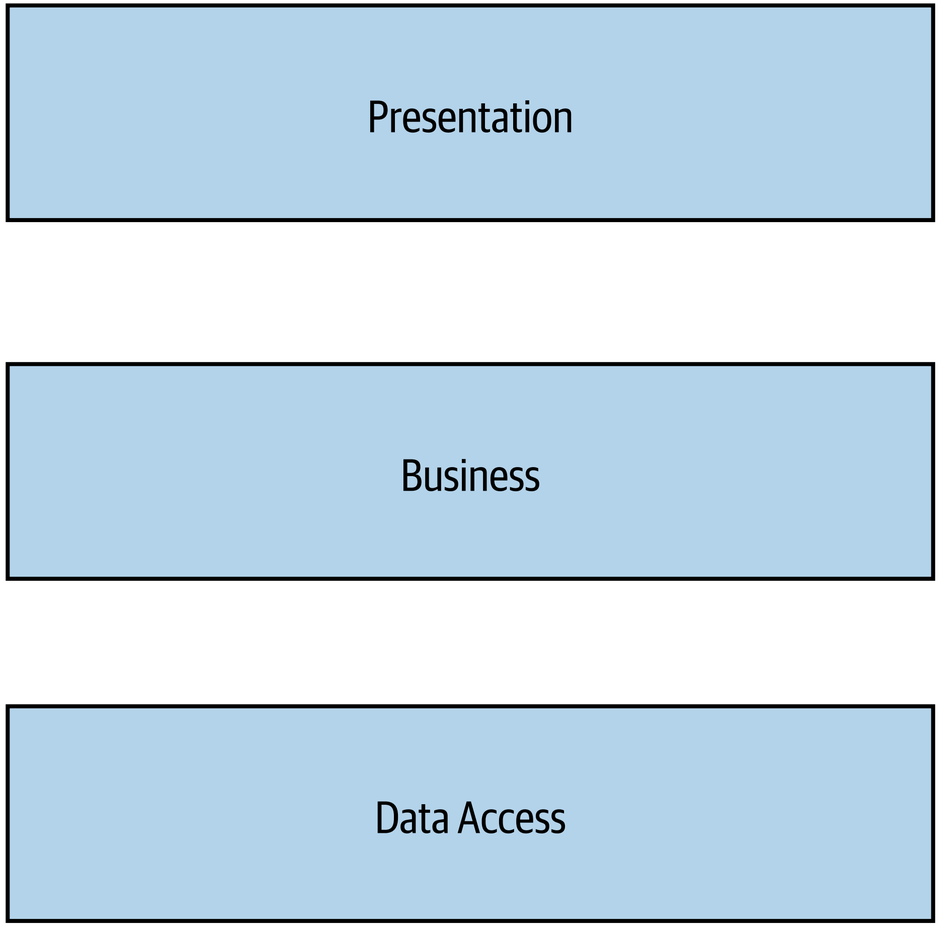
# THREE TIERED ARCHITECTURE



# THREE TIERED ARCHITECTURE

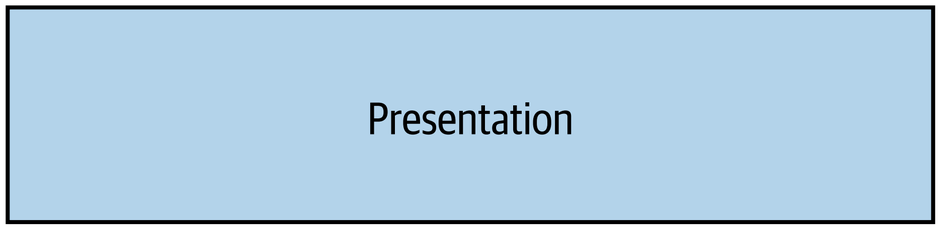


# THREE TIERED ARCHITECTURE

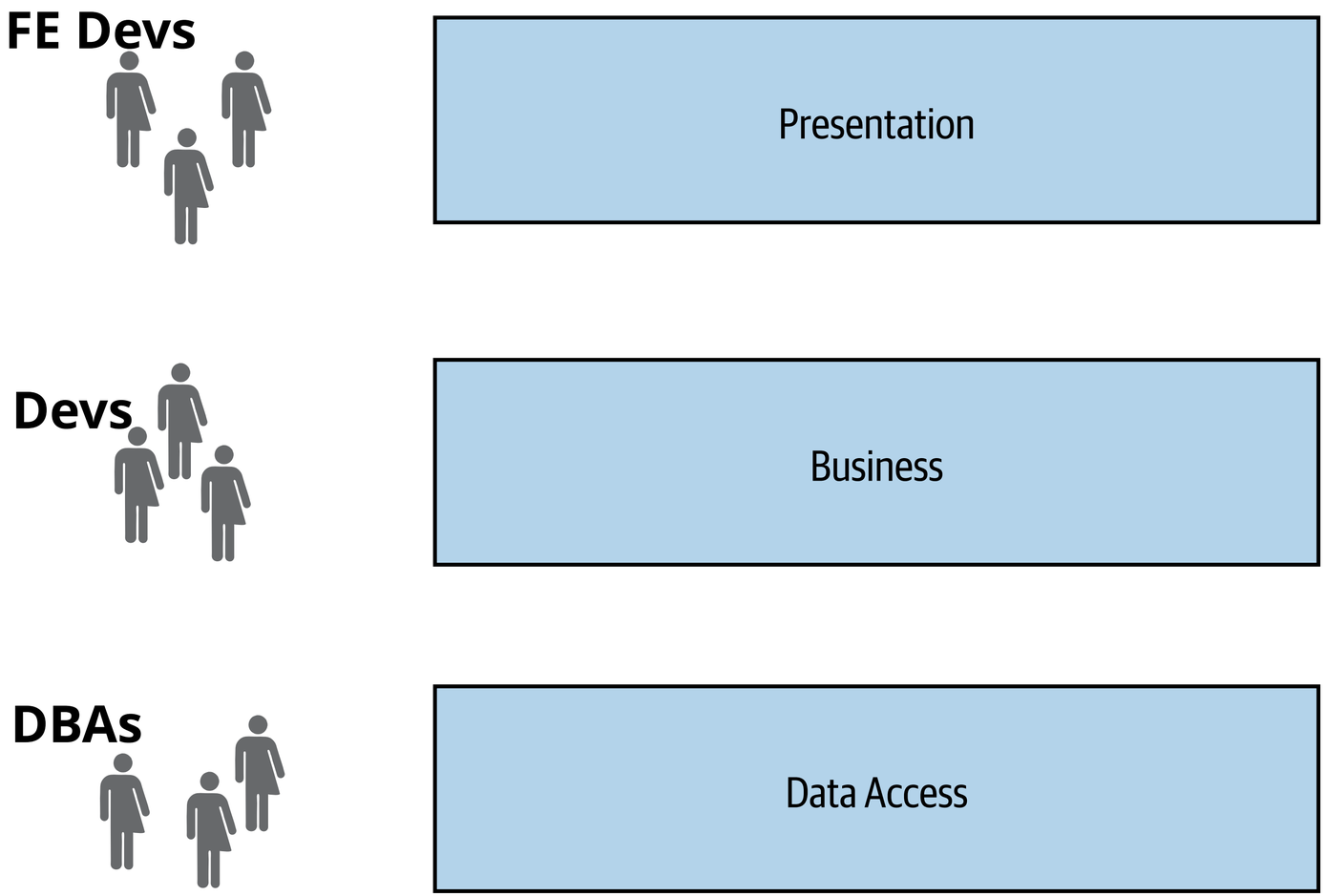
# THREE TIERED ARCHITECTURE







# THREE TIERED ARCHITECTURE

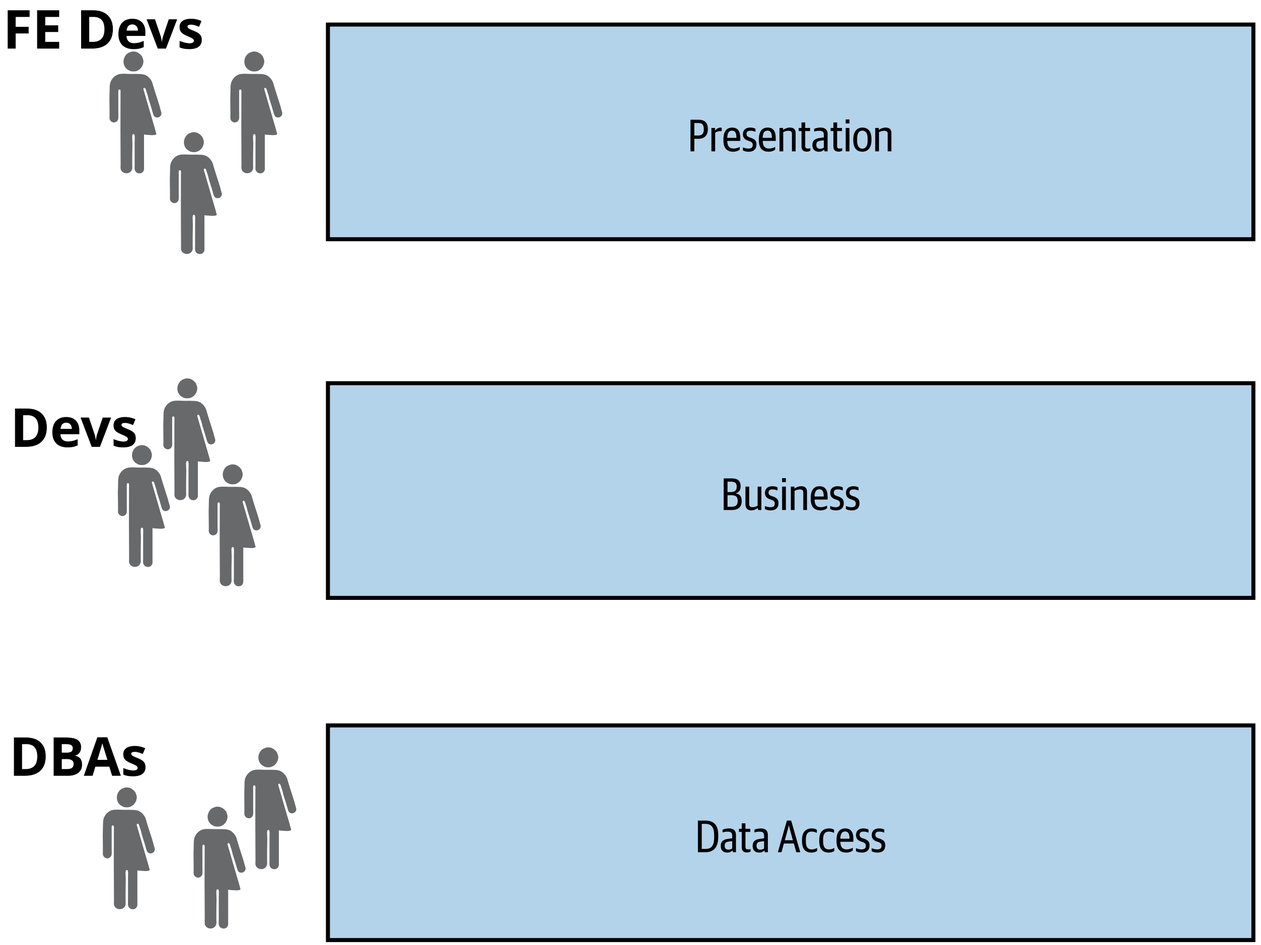


## CONWAY'S LAW

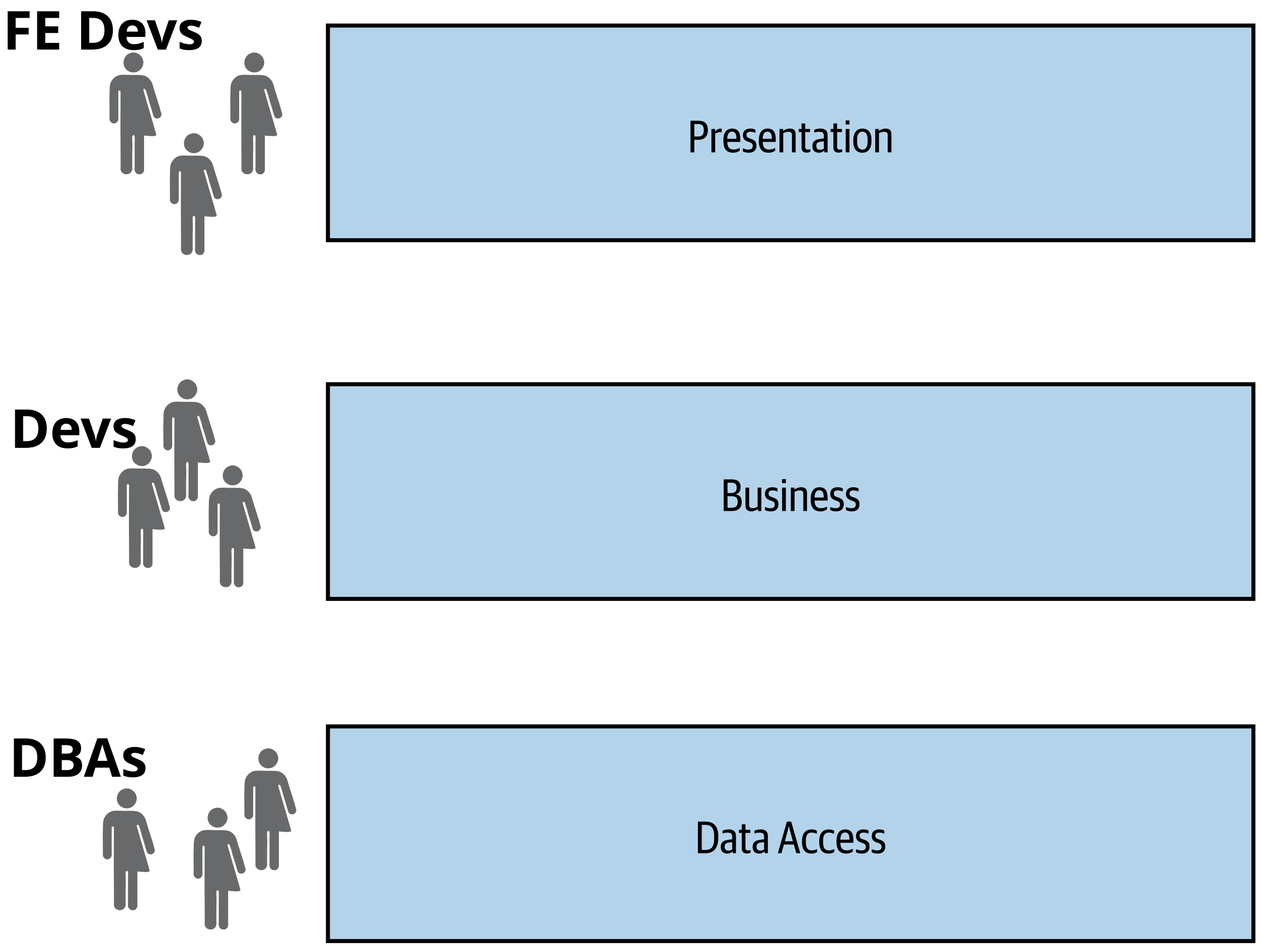
*"Organizations which design systems ... are constrained to produce designs which are copies of the communication structures of these organizations"*

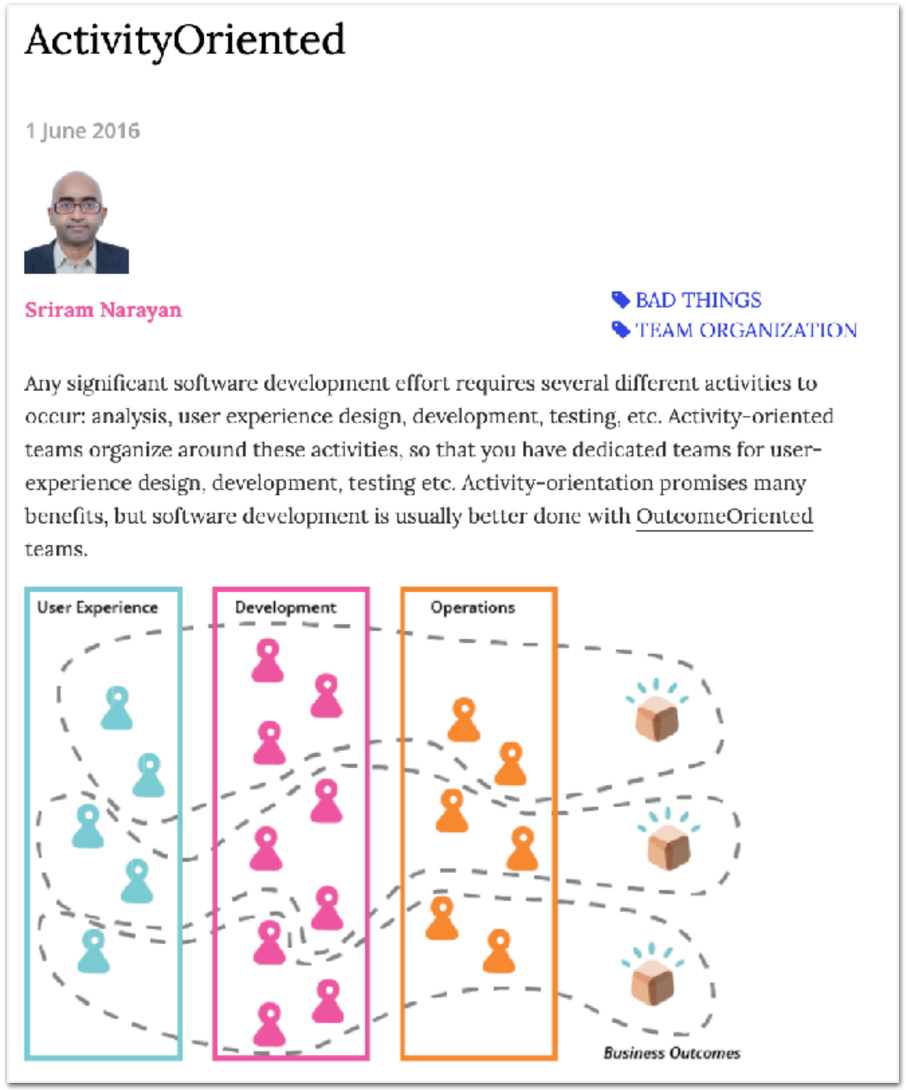
*- Melvin Conway*

# ACTIVITY ORIENTED TEAMS



# ACTIVITY ORIENTED TEAMS



ActivityOriented

1 June 2016

Sriram Narayan

BAD THINGS
TEAM ORGANIZATION

Any significant software development effort requires several different activities to occur: analysis, user experience design, development, testing, etc. Activity-oriented teams organize around these activities, so that you have dedicated teams for user-experience design, development, testing etc. Activity-orientation promises many benefits, but software development is usually better done with OutcomeOriented teams.



**https://martinfowler.com/bliki/ActivityOriented.html**

# ACTIVITY ORIENTED TEAMS

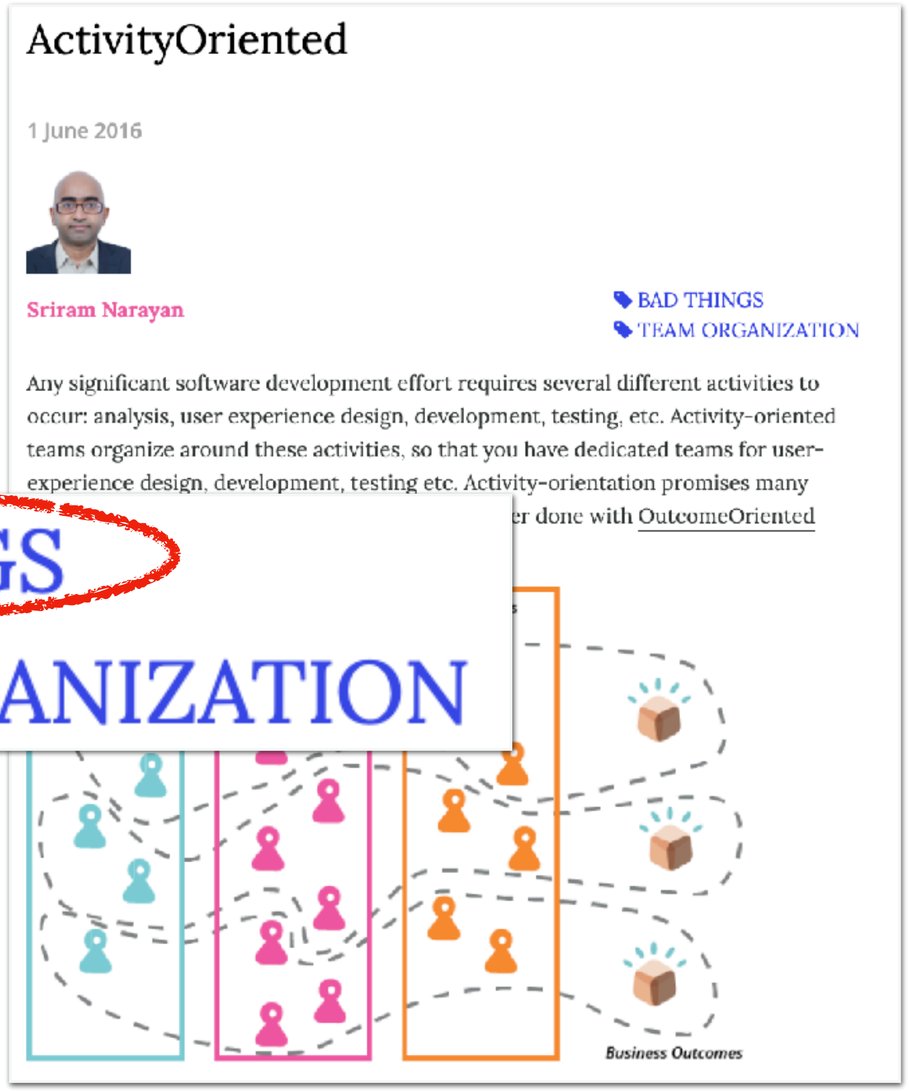
**FE Devs**

Presentation

**Devs**

**DBAs**

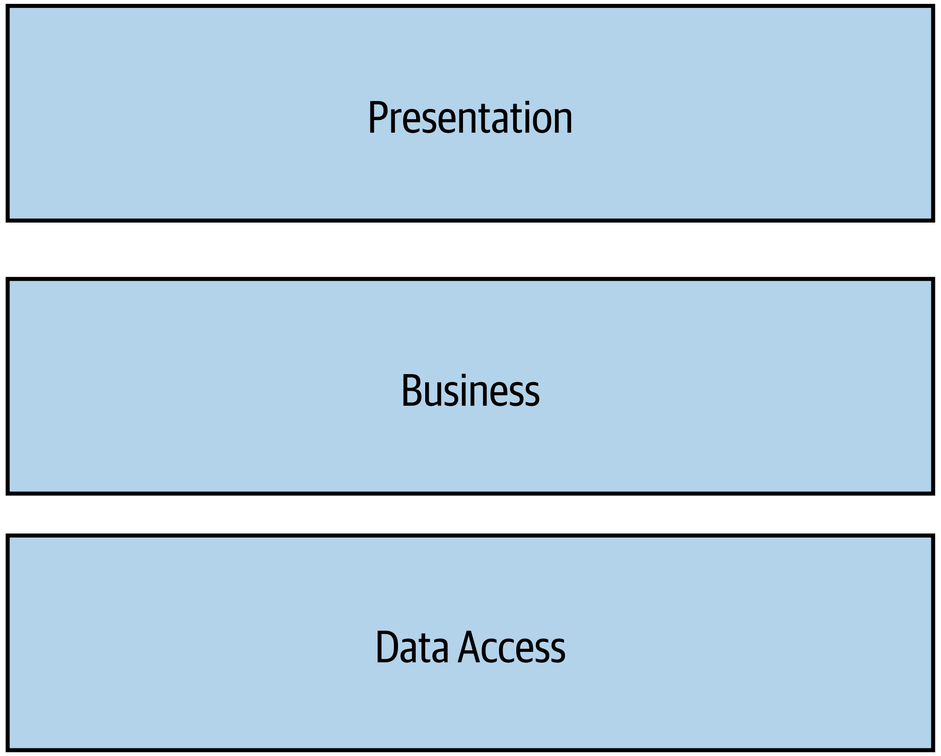
Data Access


**https://martinfowler.com/bliki/ActivityOriented.html**

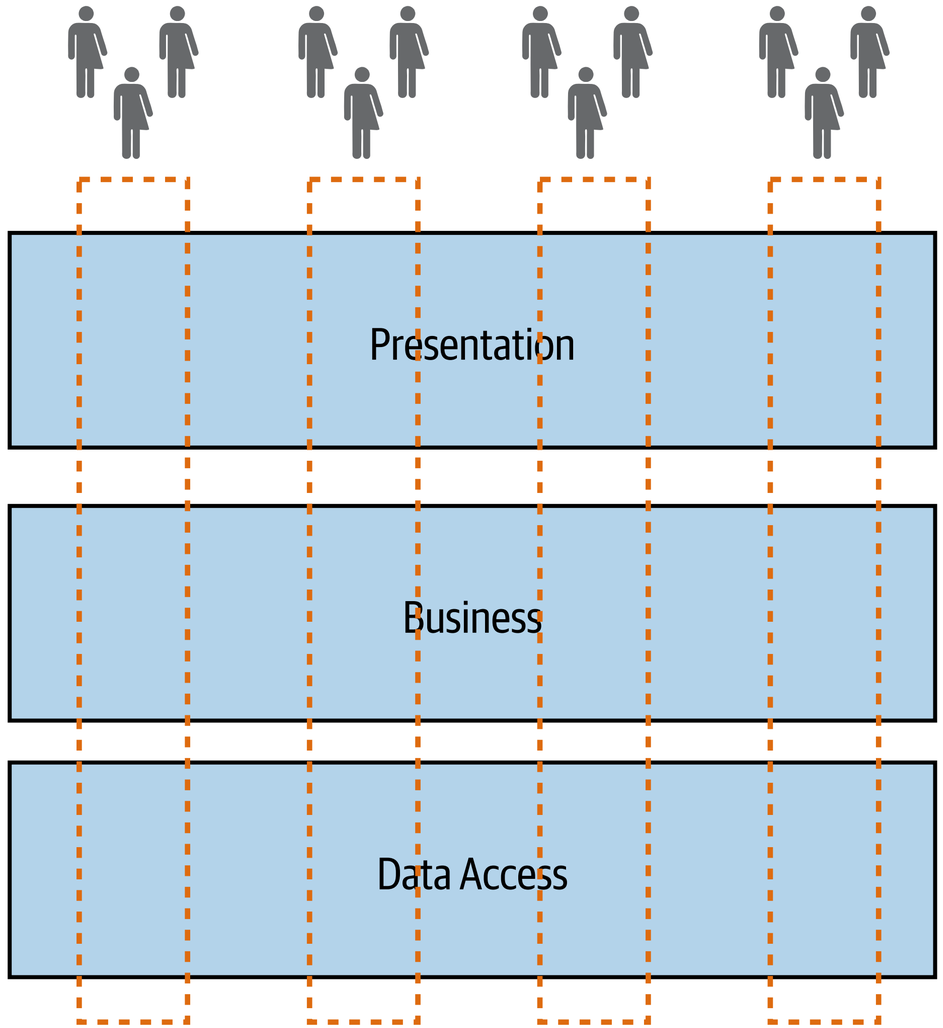
So let’s try *outcome* oriented teams!

# FEATURE-BASED TEAMS



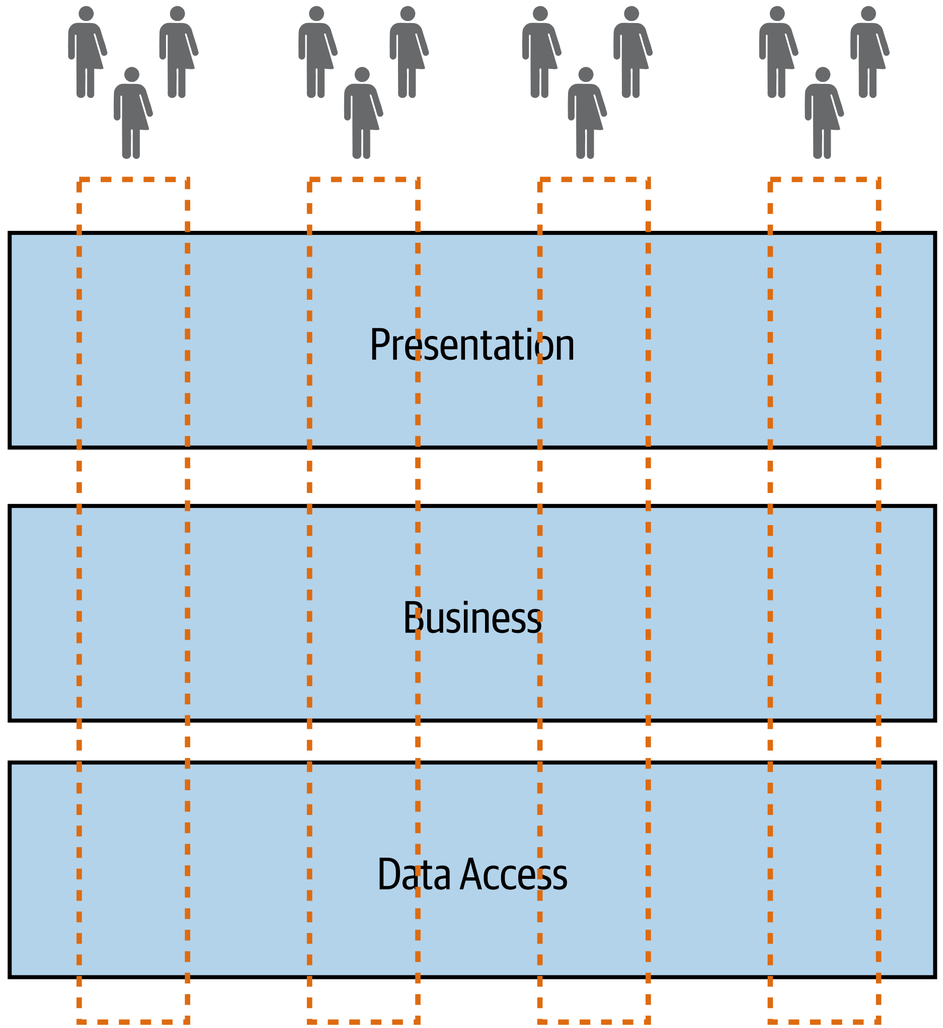

Each team “owns” delivery of a feature

# FEATURE-BASED TEAMS




Each team “owns” delivery of a feature

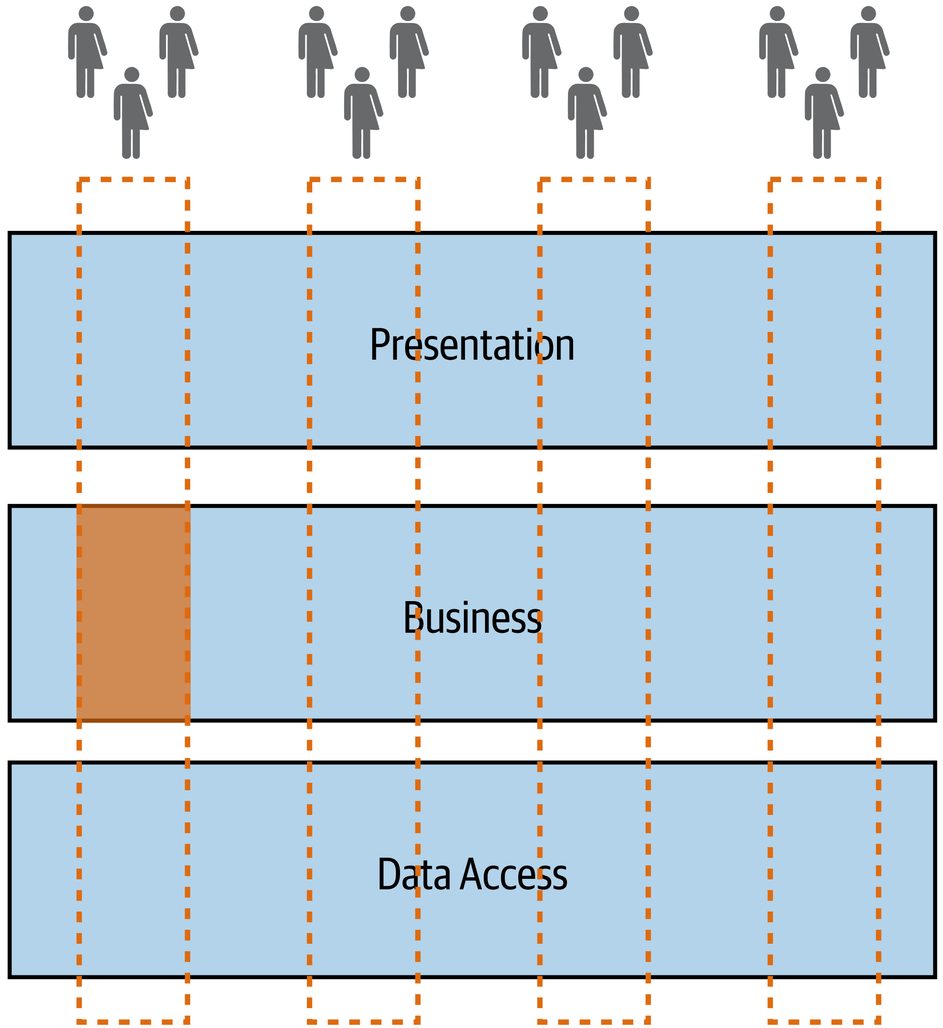
# FEATURE-BASED TEAMS




Each team “owns” delivery of a feature

But teams can still get in each others’ way as they are all working on the same underlying things

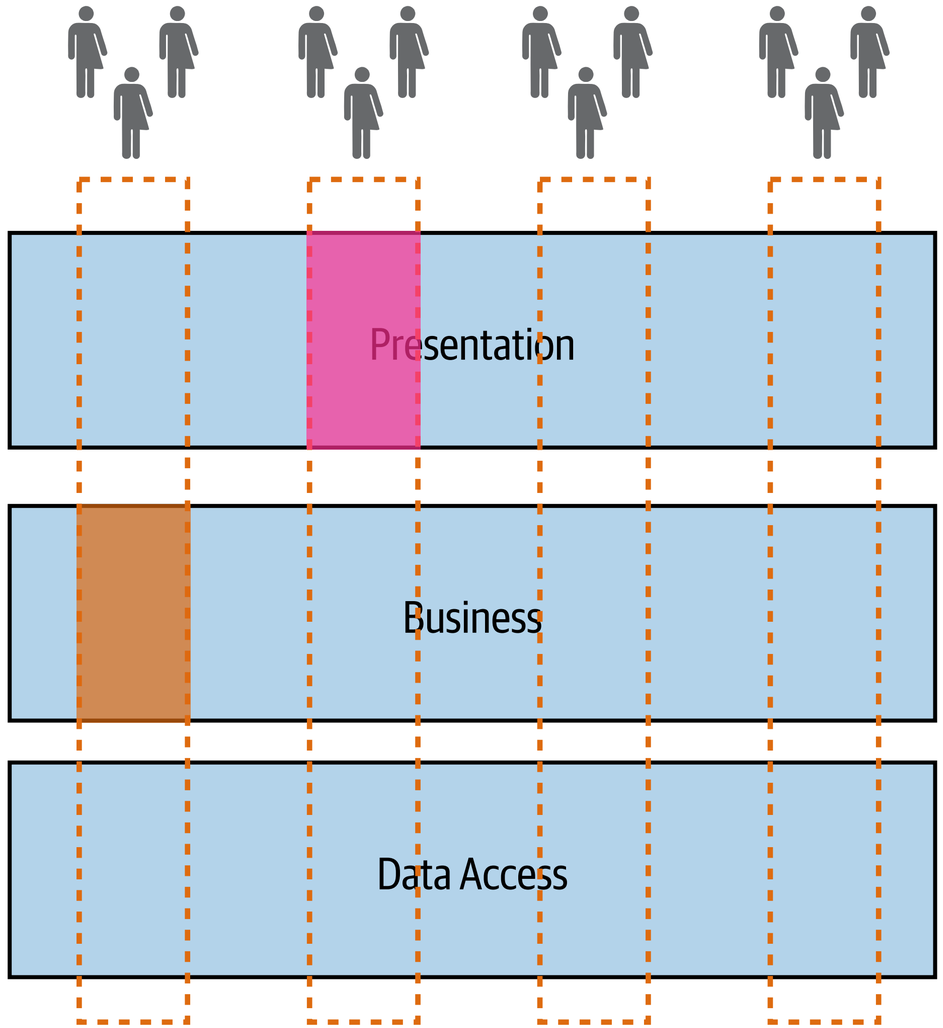
# FEATURE-BASED TEAMS




Each team “owns” delivery of a feature

But teams can still get in each others’ way as they are all working on the same underlying things

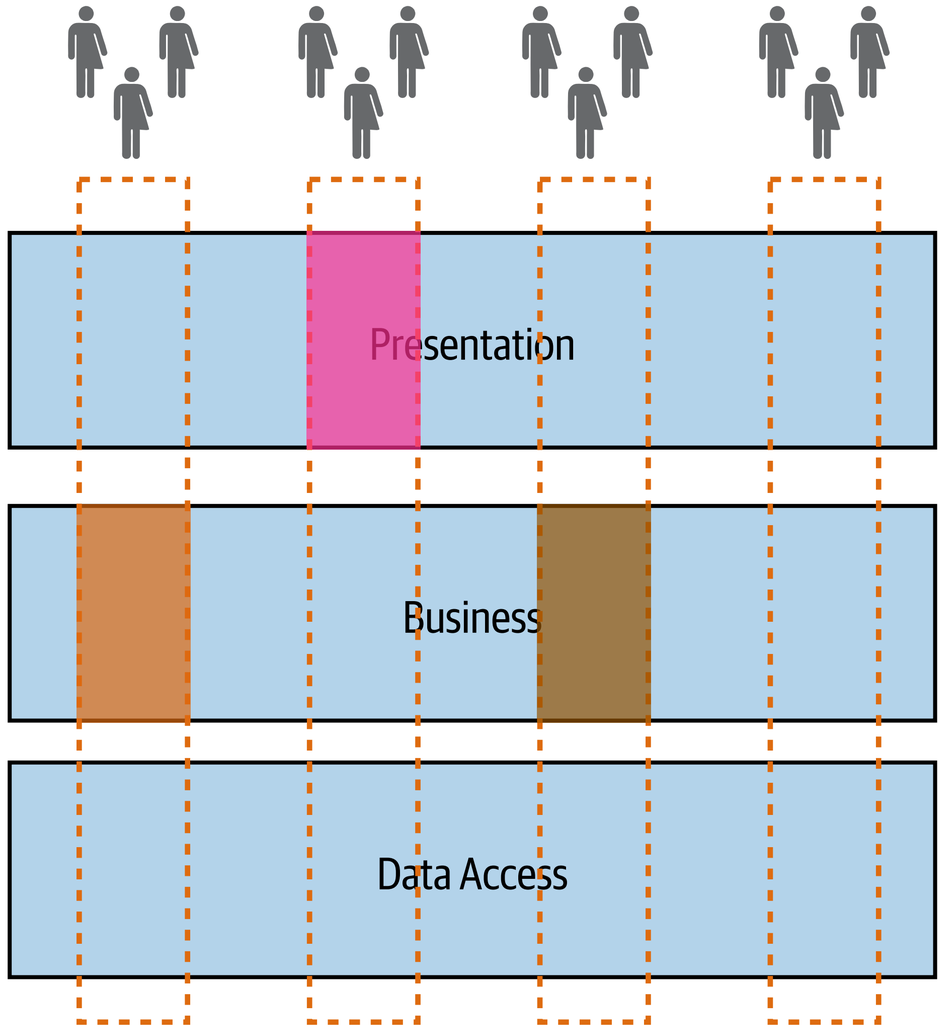
# FEATURE-BASED TEAMS




Each team “owns” delivery of a feature

But teams can still get in each others’ way as they are all working on the same underlying things

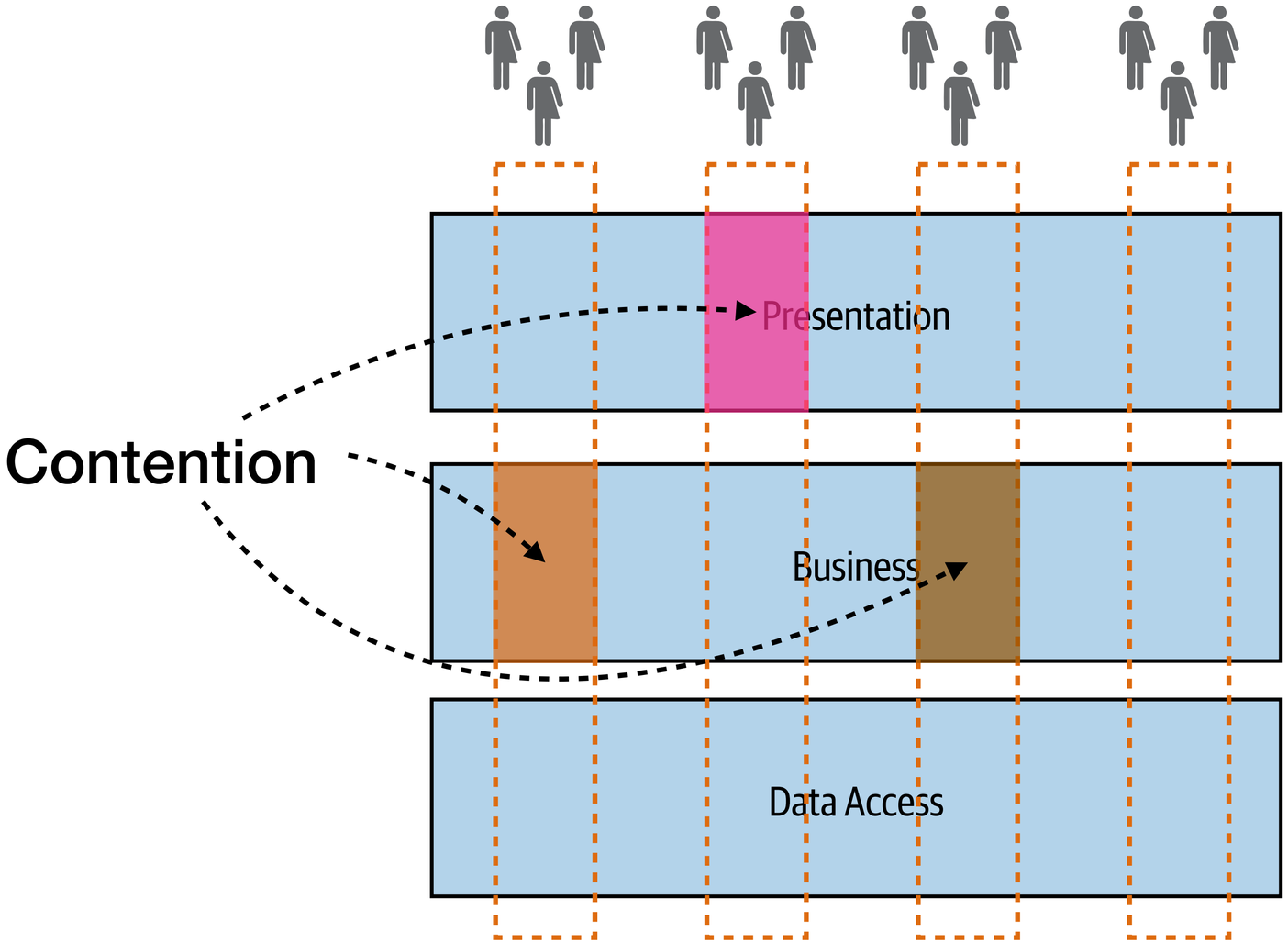
# FEATURE-BASED TEAMS




Each team “owns” delivery of a feature

But teams can still get in each others’ way as they are all working on the same underlying things

# FEATURE-BASED TEAMS

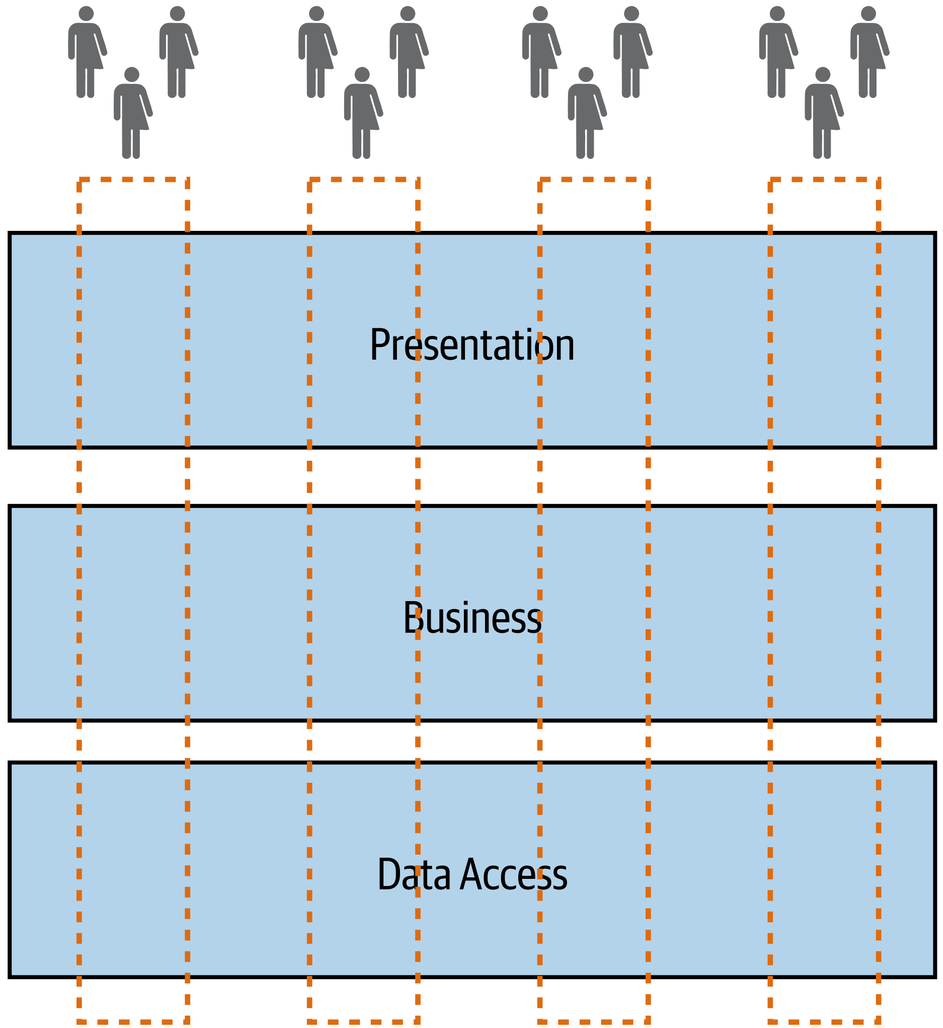



Each team “owns” delivery of a feature

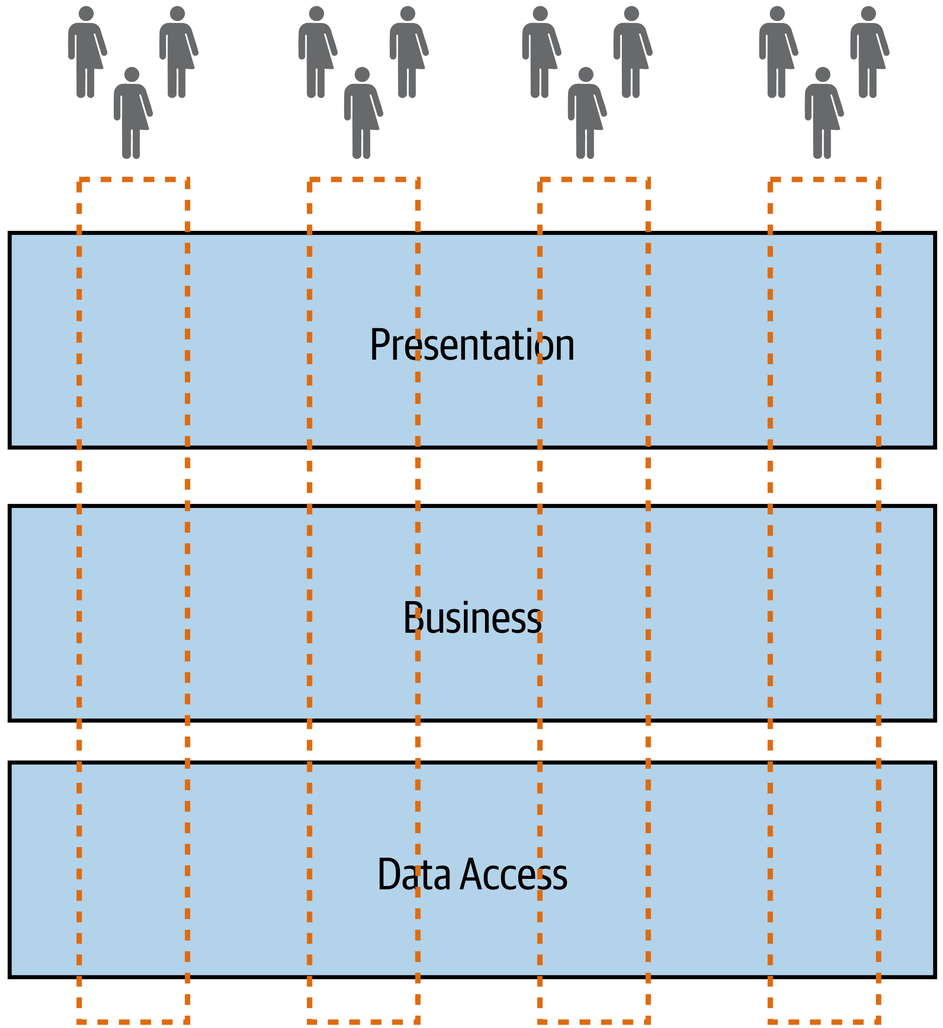
But teams can still get in each others’ way as they are all working on the same underlying things

So better, but not perfect

# FEATURE-BASED TEAMS

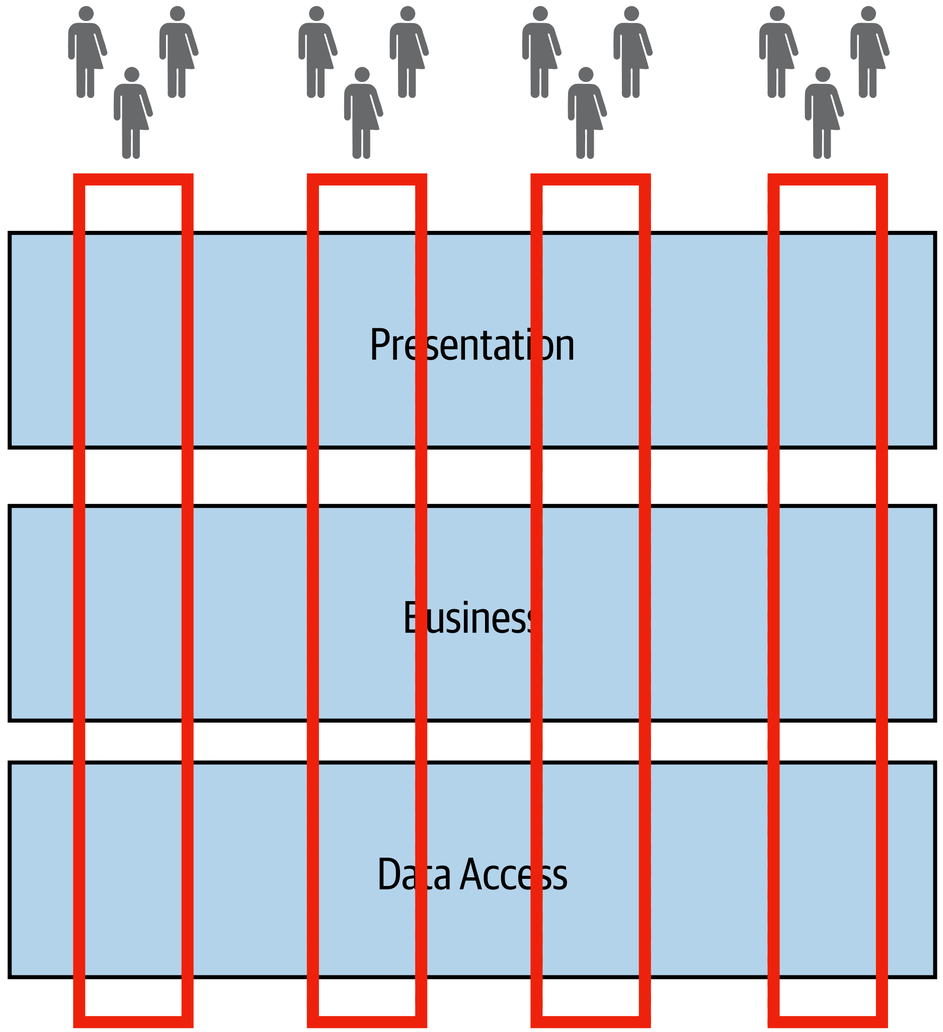


# FEATURE-BASED TEAMS



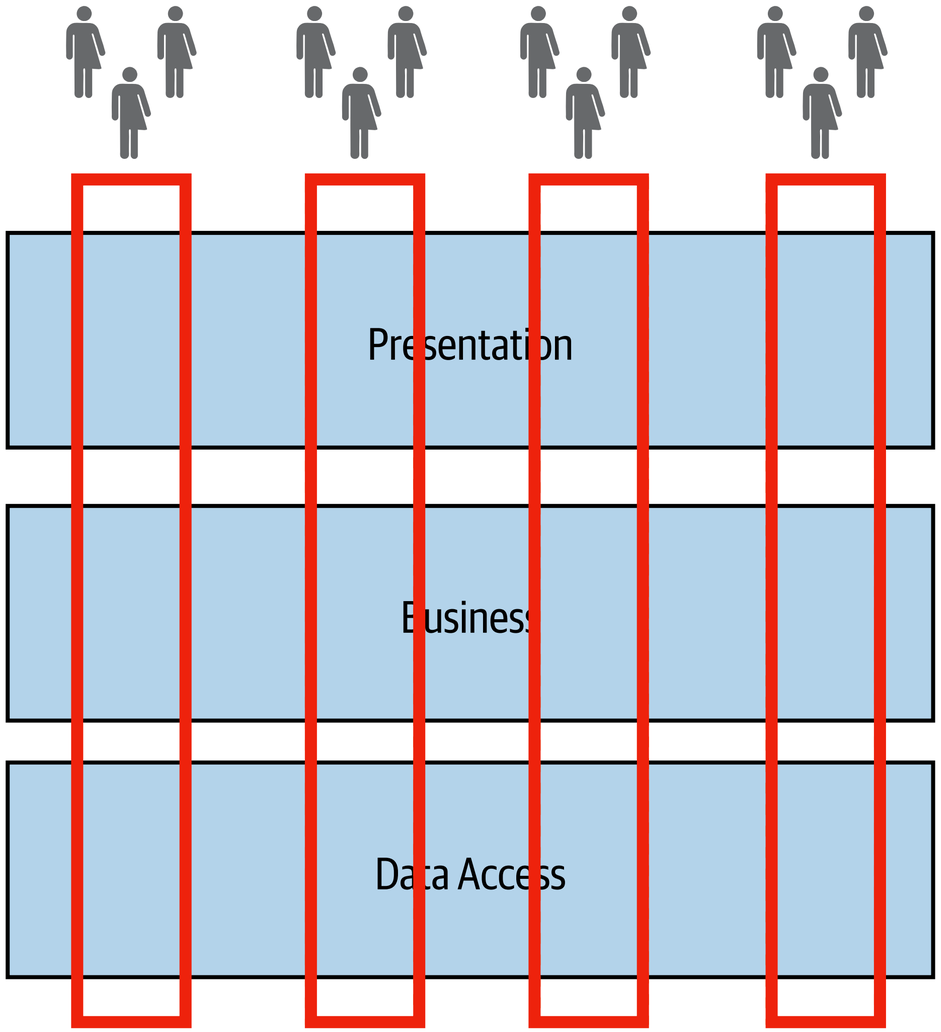

**We'd like to give each team full ownership over their assets**

# FEATURE-BASED TEAMS




We'd like to give each team full ownership over their assets

# FEATURE-BASED TEAMS




We'd like to give each team full ownership over their assets

Reduce contention, improve autonomy

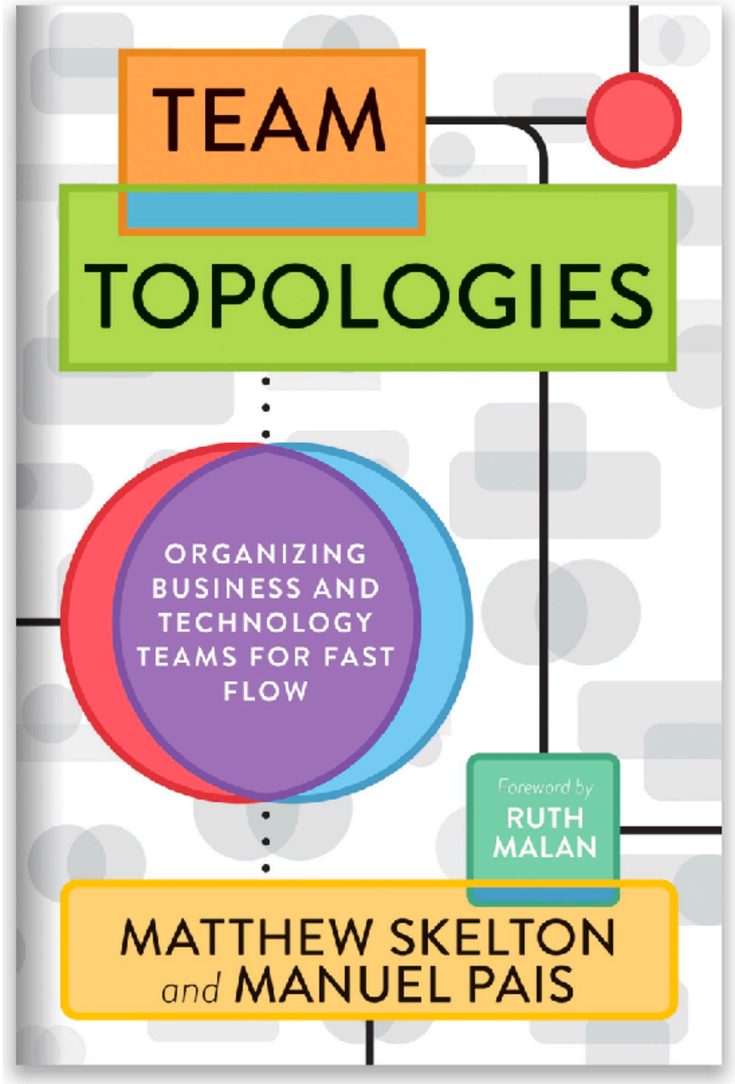
# POLL: WHAT TYPE OF OWNERSHIP MODEL DO YOU HAVE?

Any developer can change any microservice whenever they want

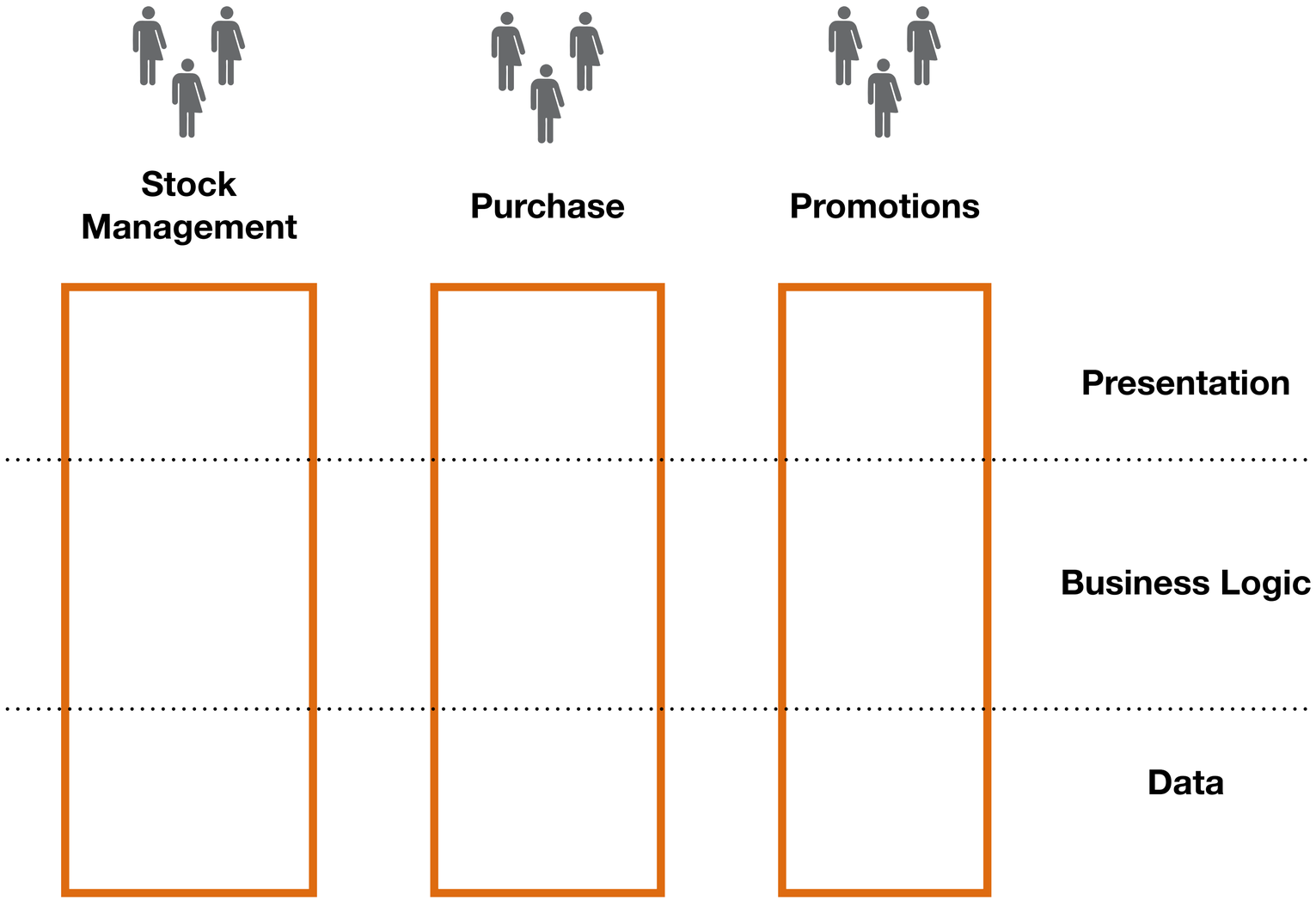
Some microservices are shared, some are owned by specific teams

Each microservice is owned by a dedicated team

# TEAM TOPOLOGIES

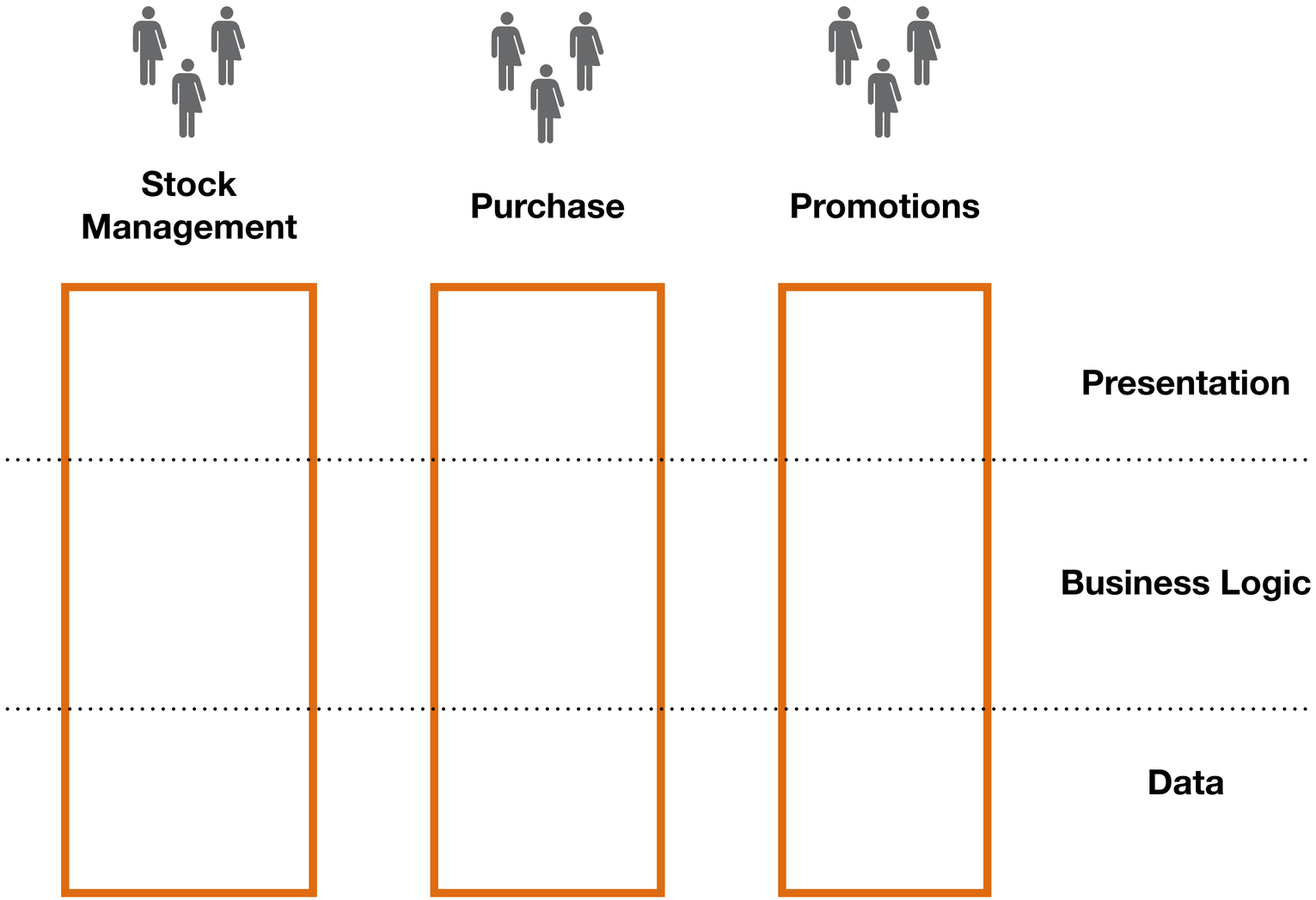


# STREAM-ALIGNED TEAMS



# STREAM-ALIGNED TEAMS

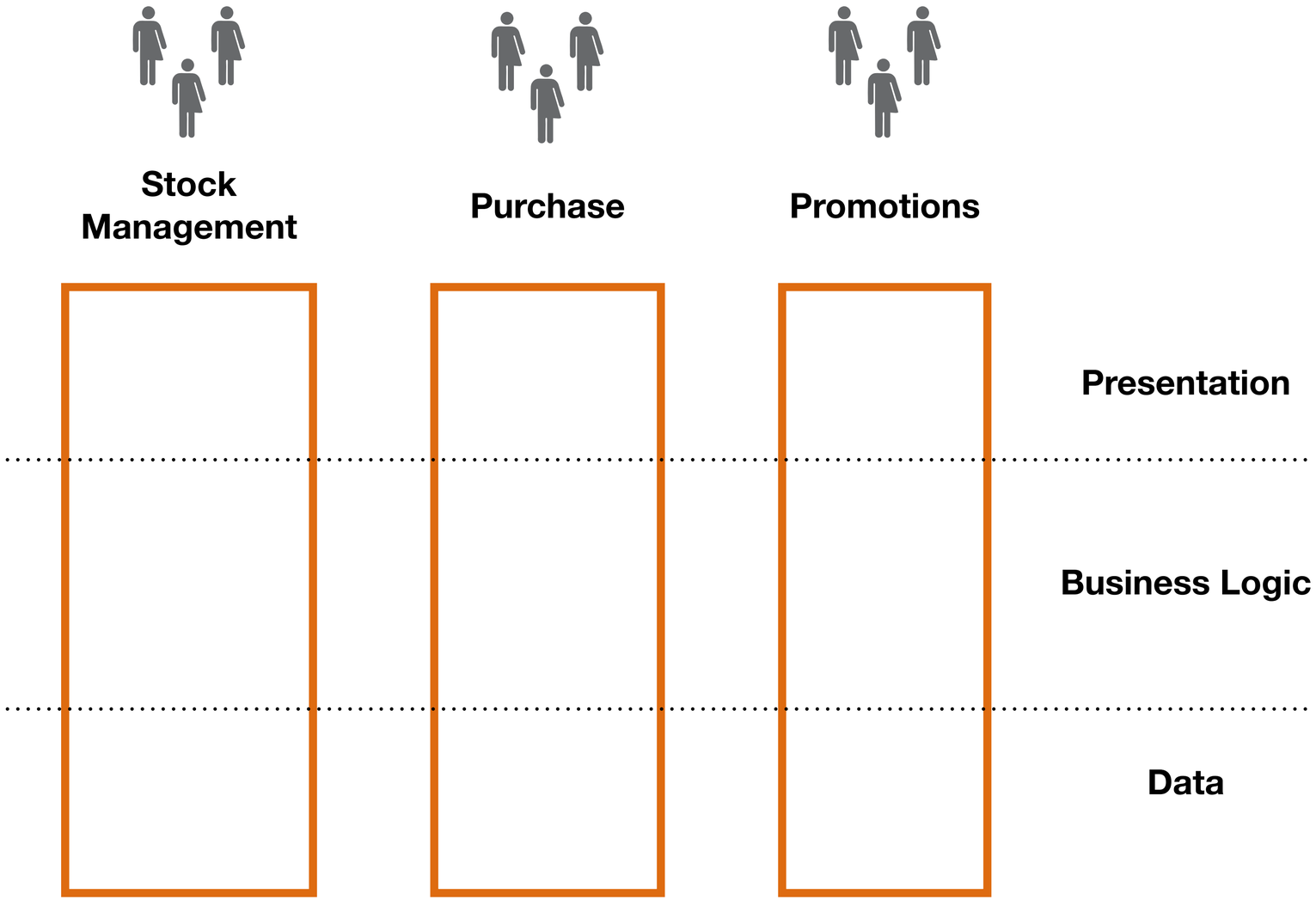
Focused on a valuable stream of work



# STREAM-ALIGNED TEAMS

Focused on a valuable stream of work

Long-lived “product” oriented rather than project oriented

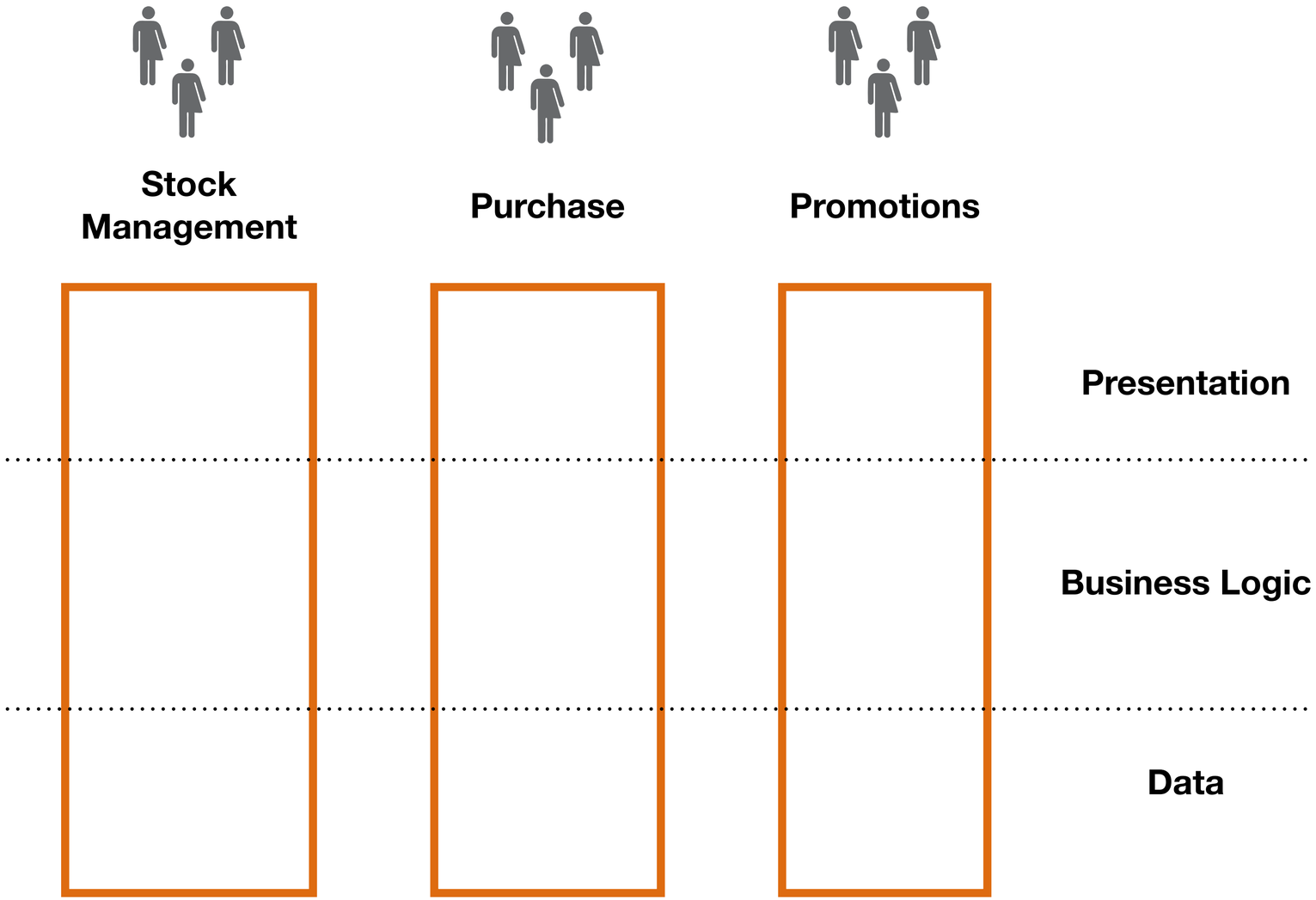


# STREAM-ALIGNED TEAMS

Focused on a valuable stream of work

Long-lived “product” oriented rather than project oriented

Has ownership of their assets

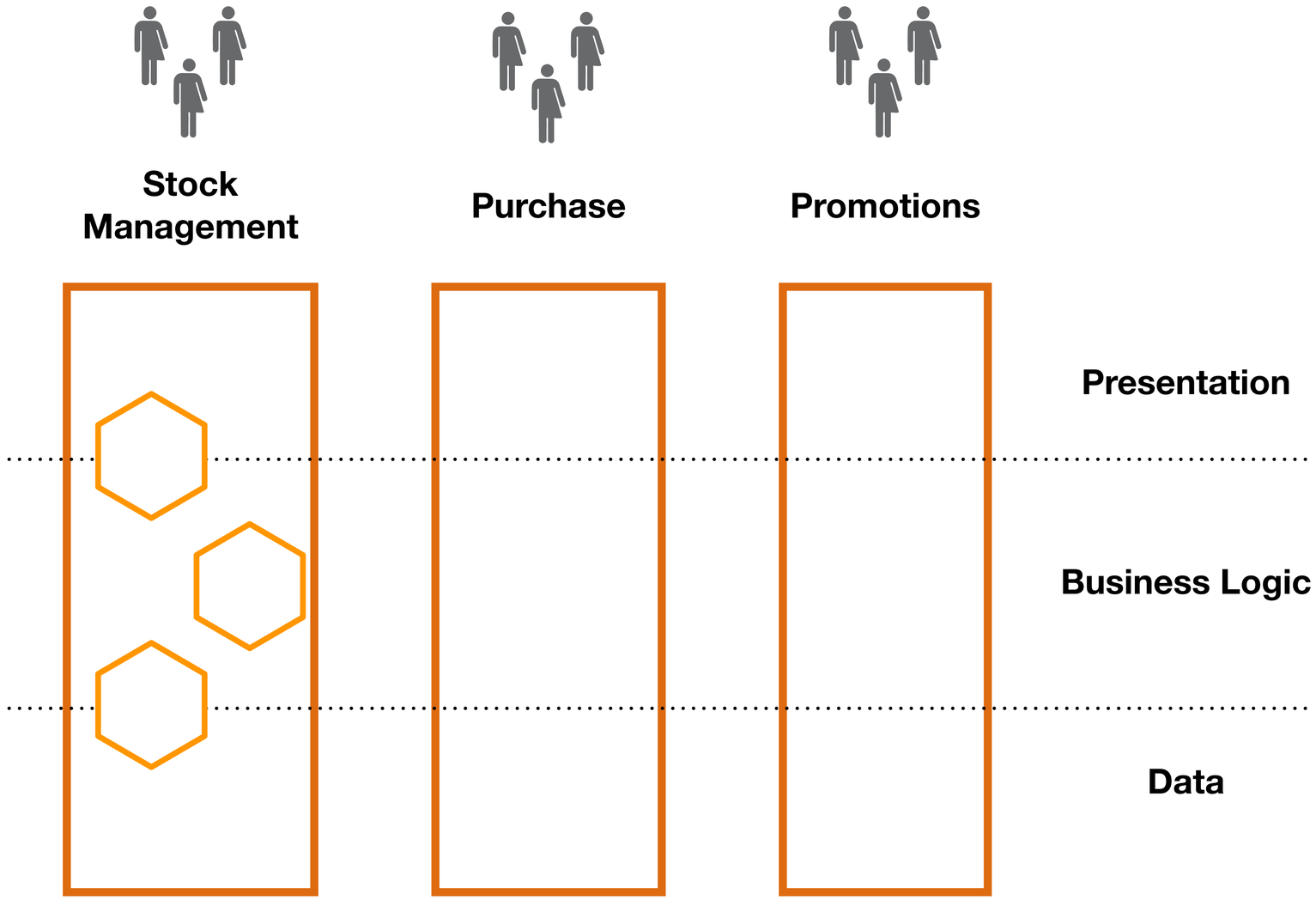


# STREAM-ALIGNED TEAMS

Focused on a valuable stream of work

Long-lived “product” oriented rather than project oriented

Has ownership of their assets

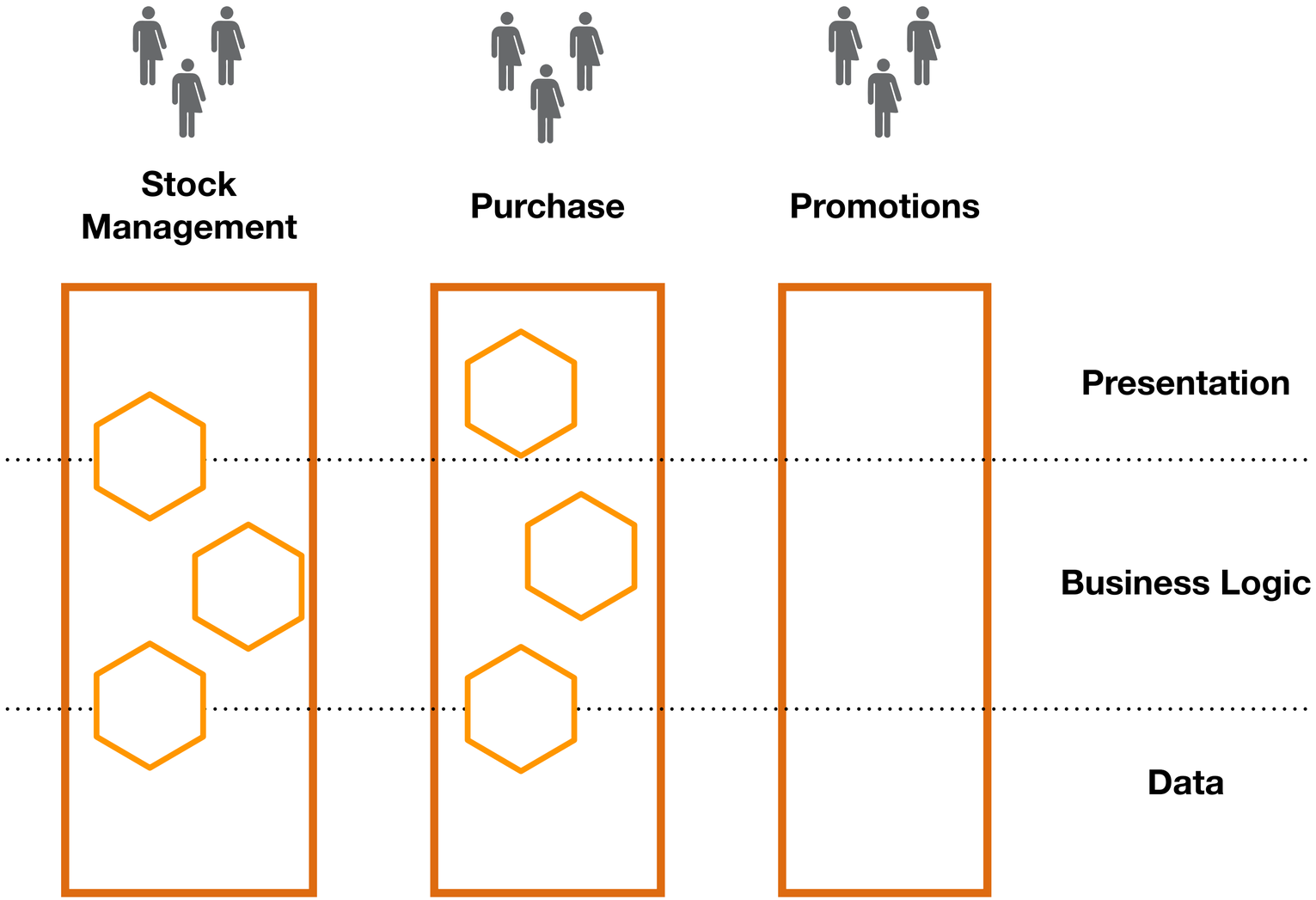


# STREAM-ALIGNED TEAMS

Focused on a valuable stream of work

Long-lived “product” oriented rather than project oriented

Has ownership of their assets

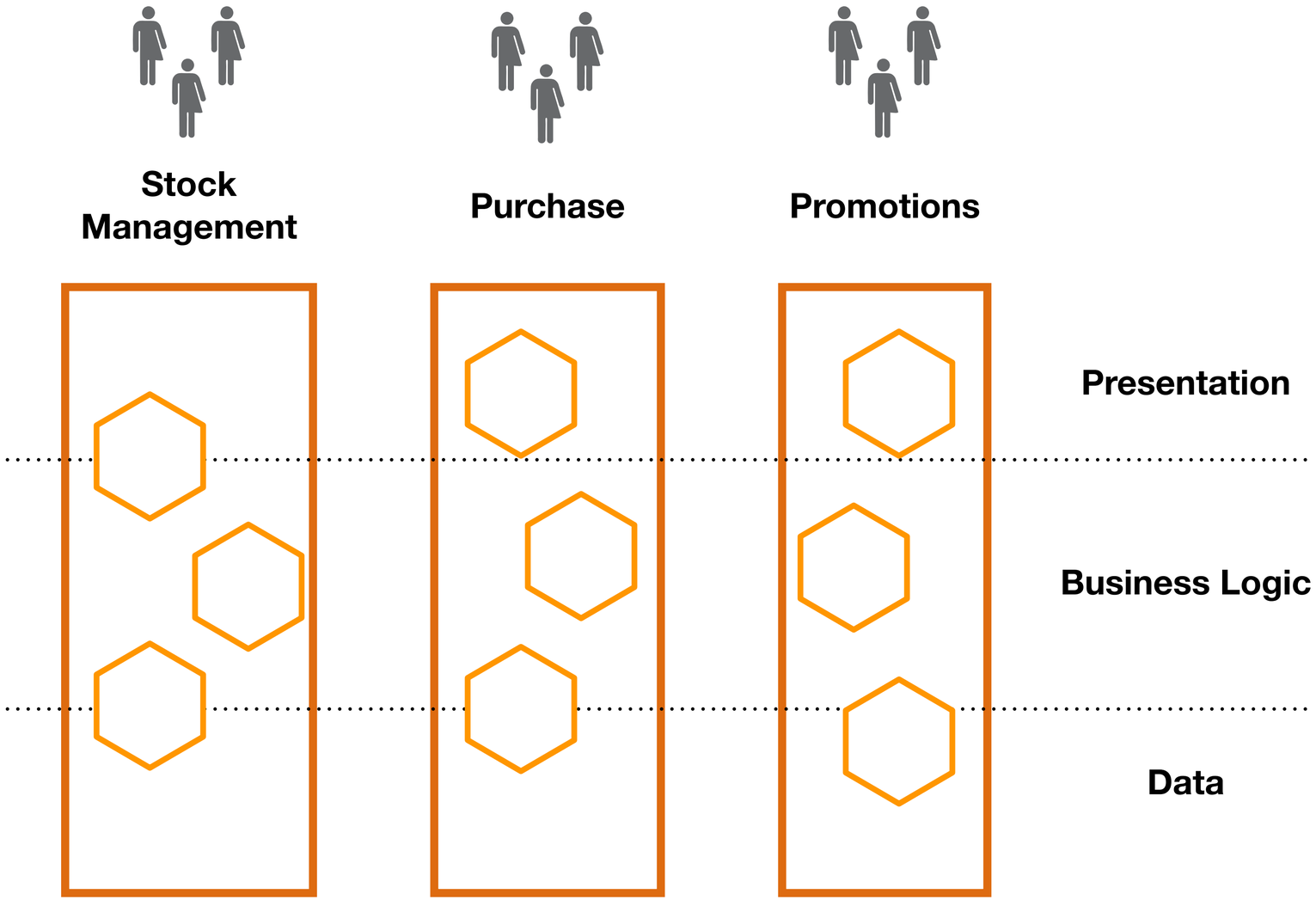


# STREAM-ALIGNED TEAMS

Focused on a valuable stream of work

Long-lived “product” oriented rather than project oriented

Has ownership of their assets

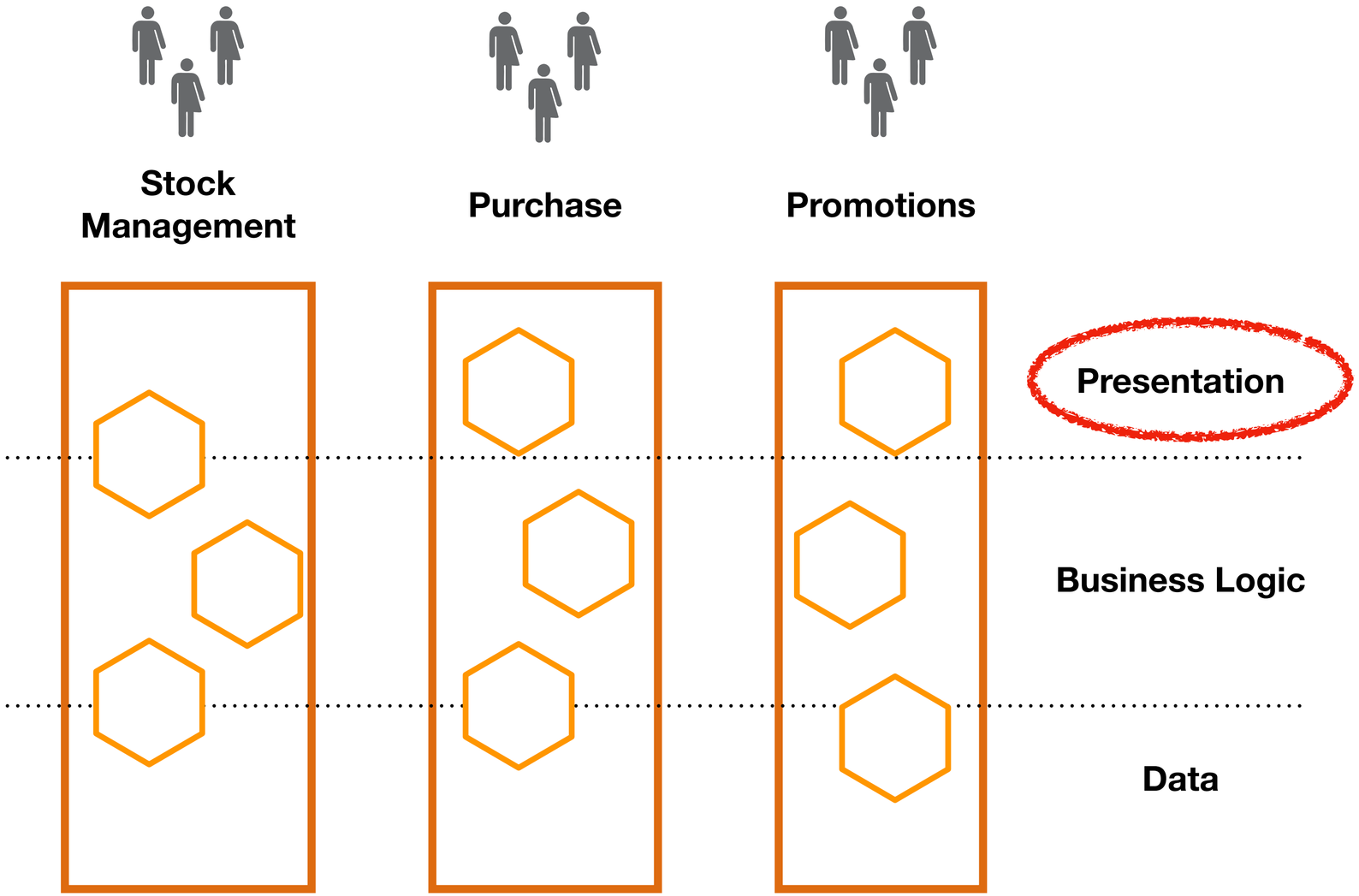


# STREAM-ALIGNED TEAMS

Focused on a valuable stream of work

Long-lived “product” oriented rather than project oriented

Has ownership of their assets



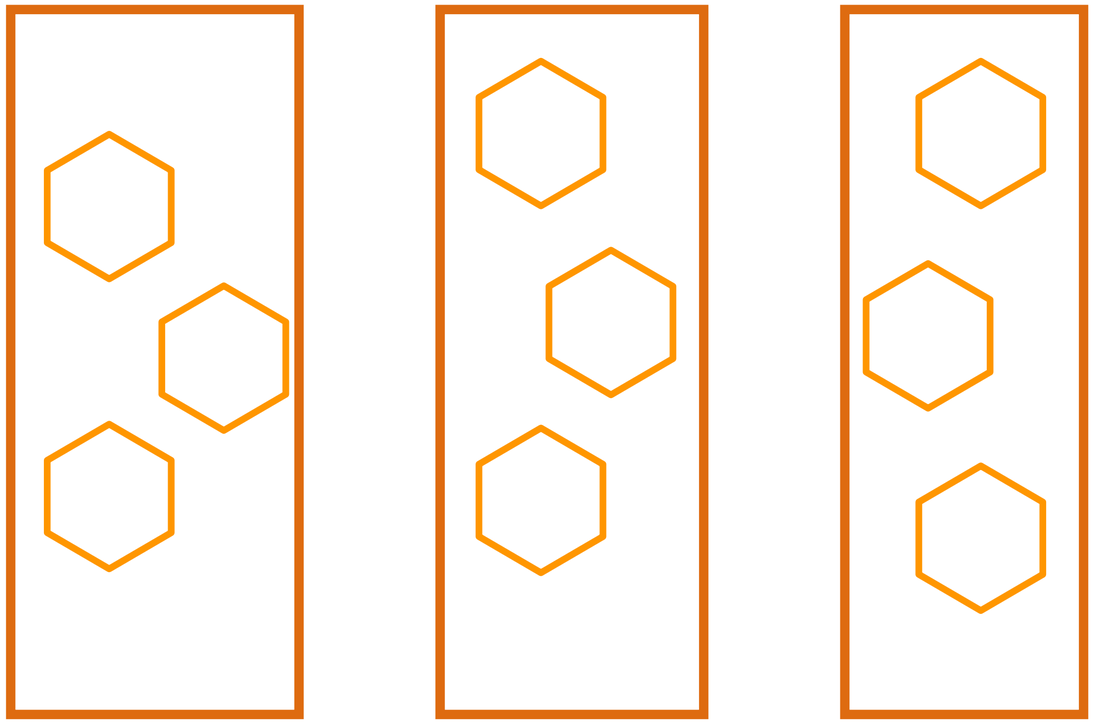
# POLL: DO YOU HAVE A SEPARATE UI LAYER?

Yes we do - separate backend and frontend teams

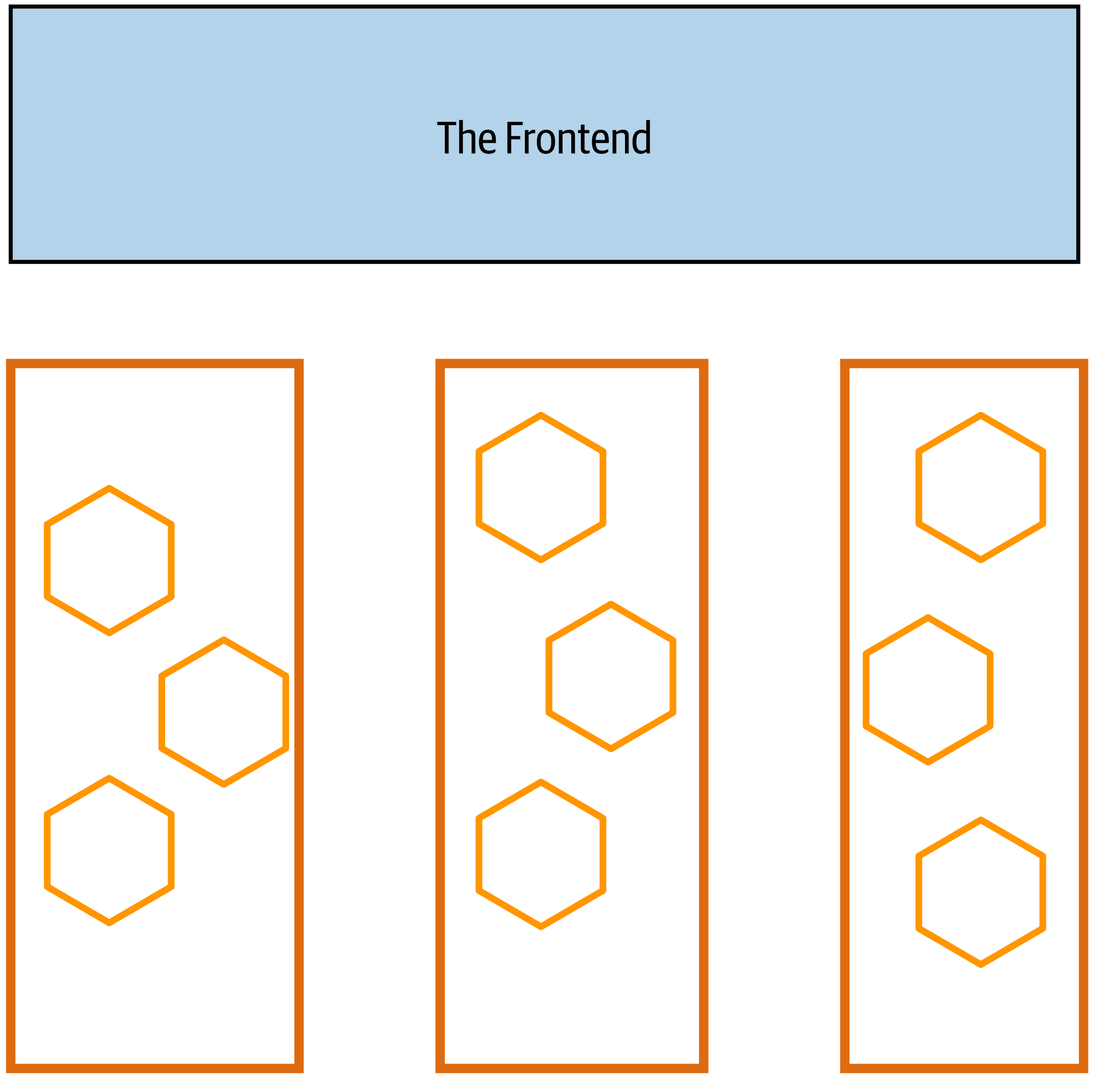
No, teams own their own frontend and backend

We don’t have a frontend!

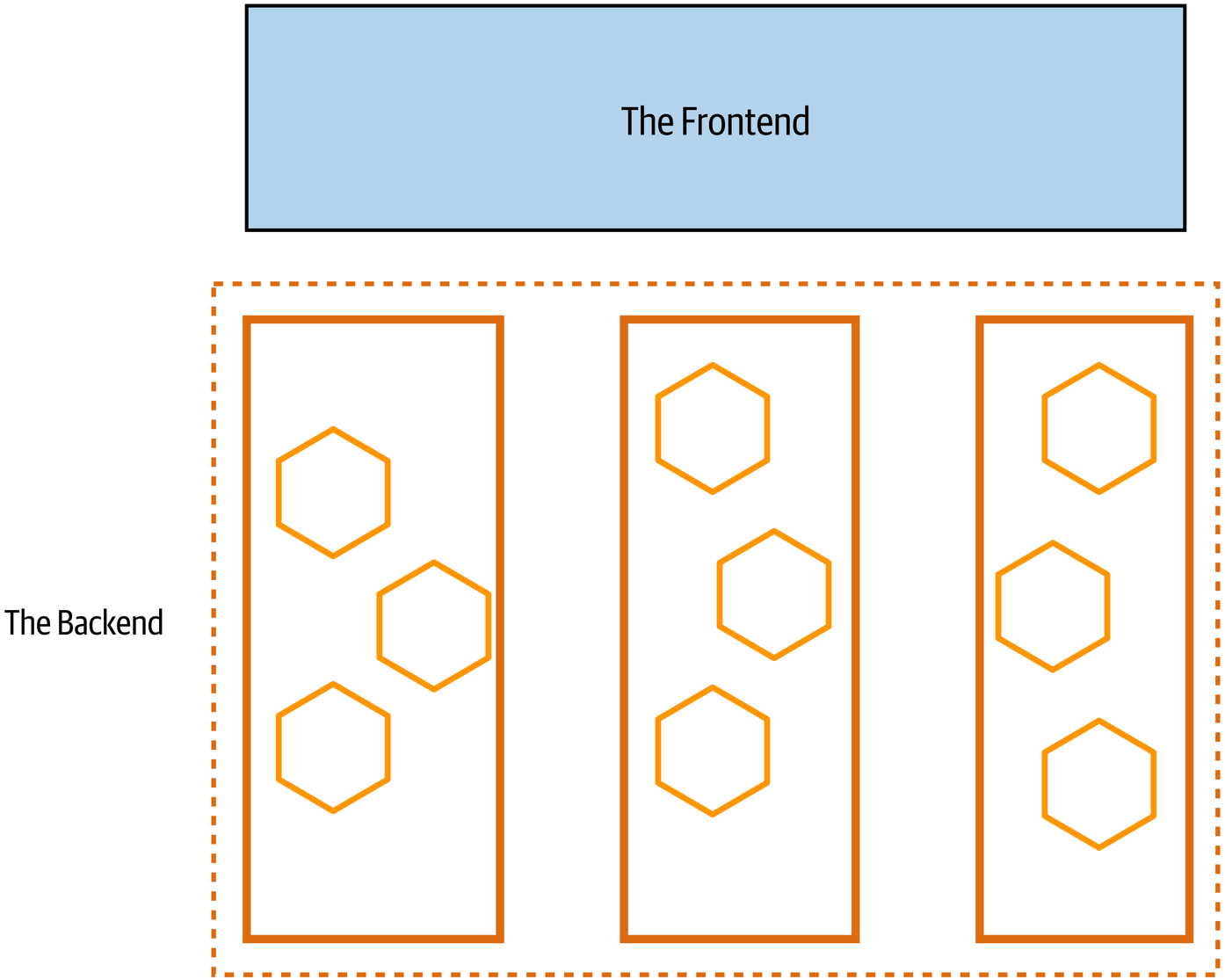
# THE UI?



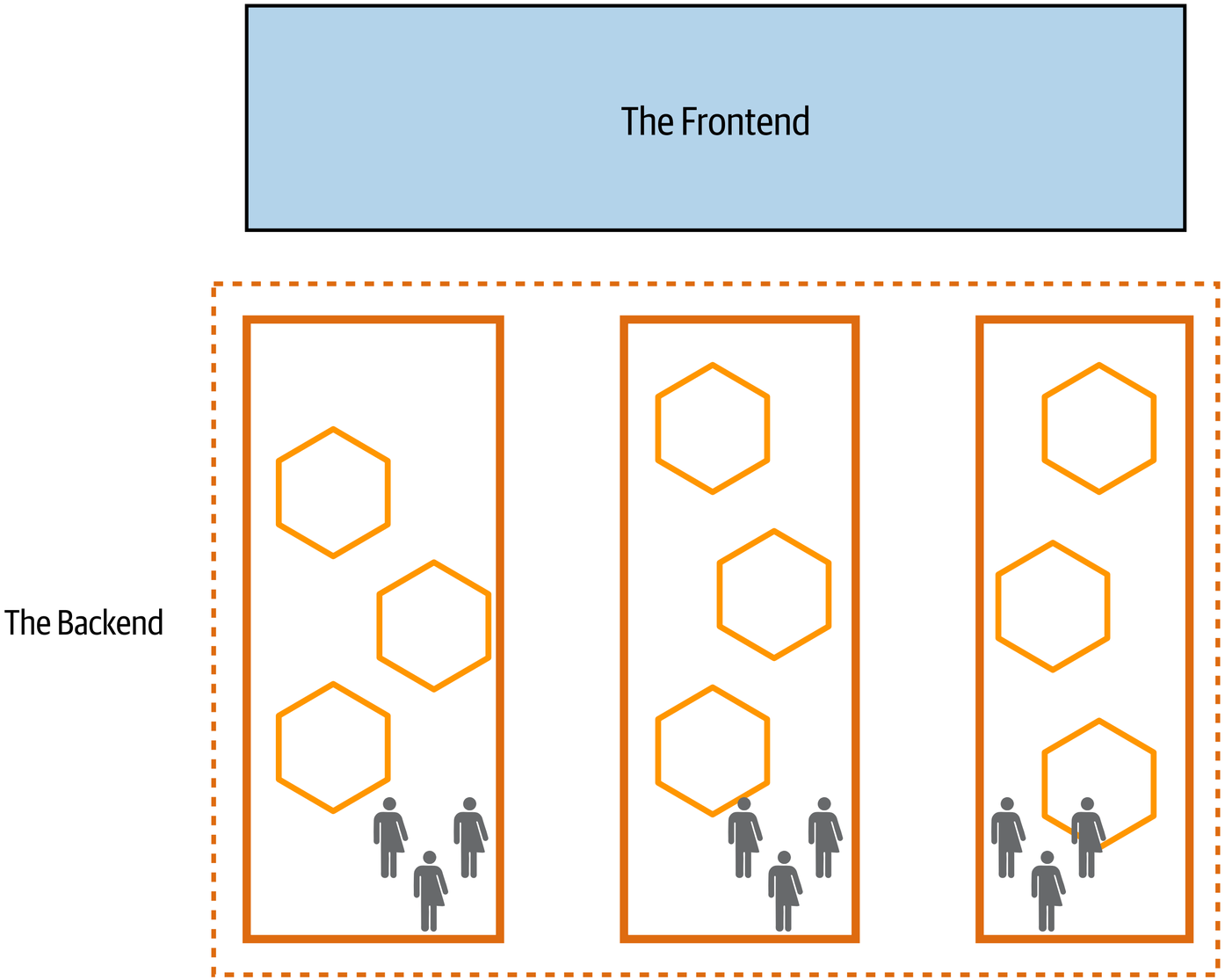
# THE UI?



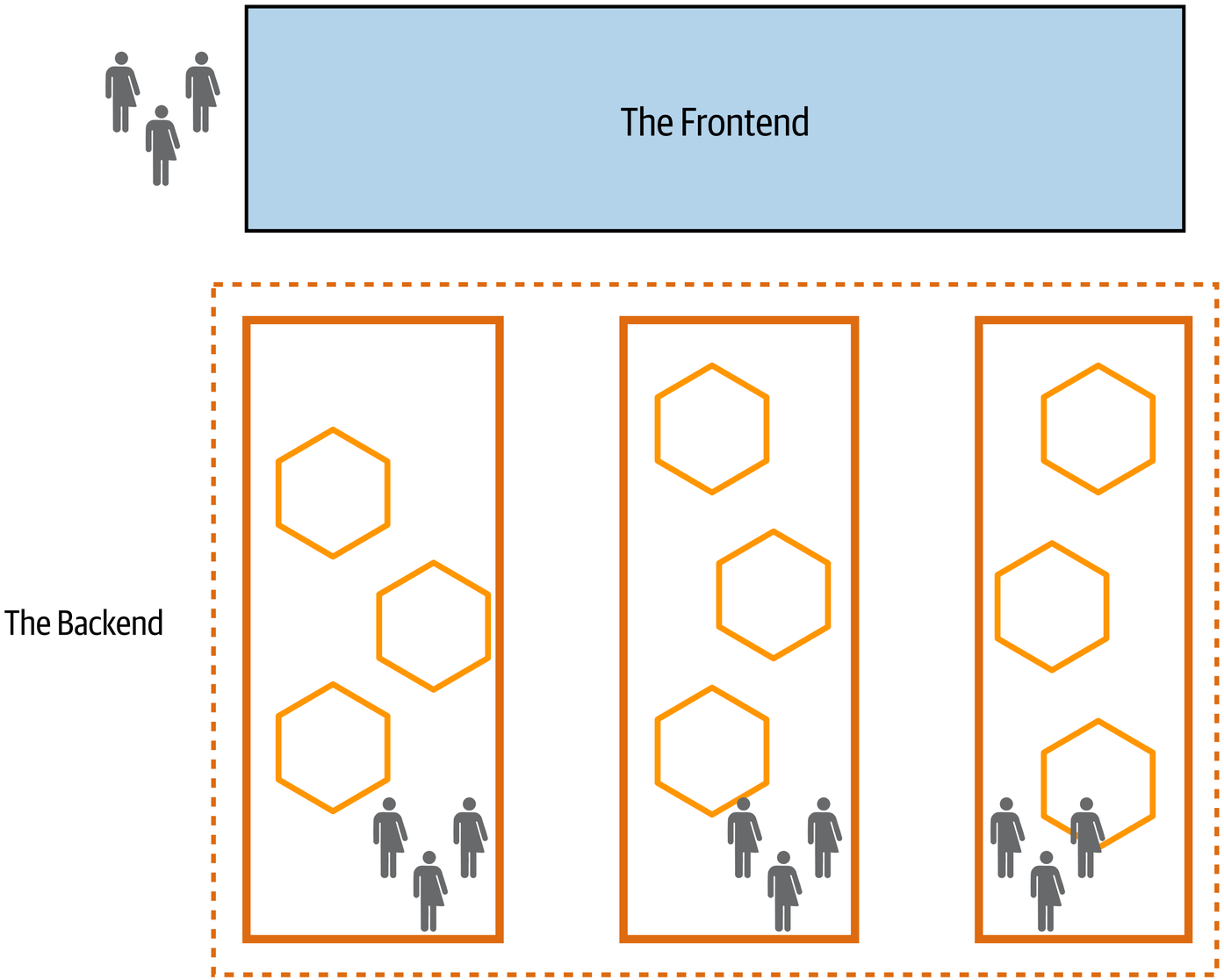
# THE UI?



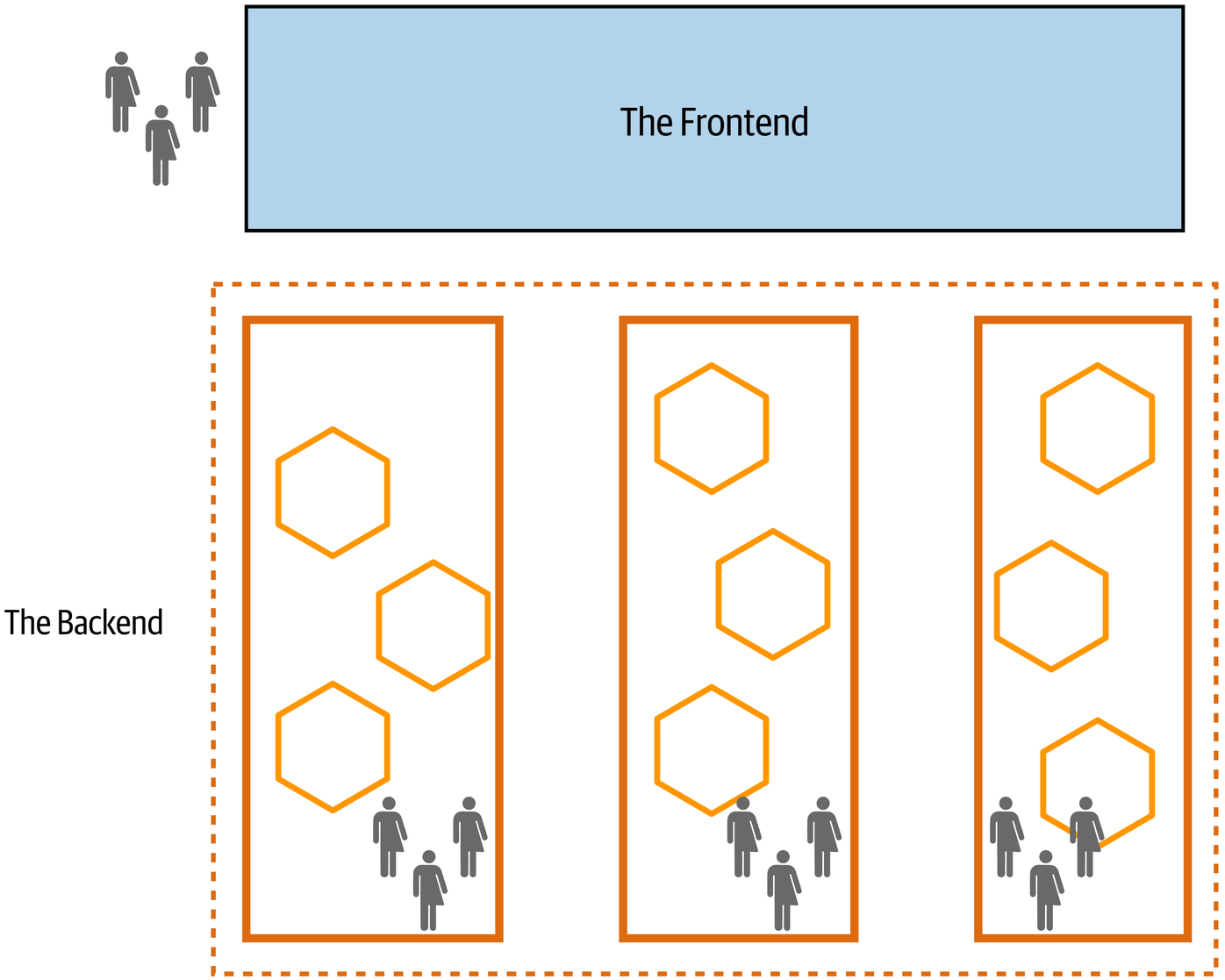
# THE UI?



# THE UI?

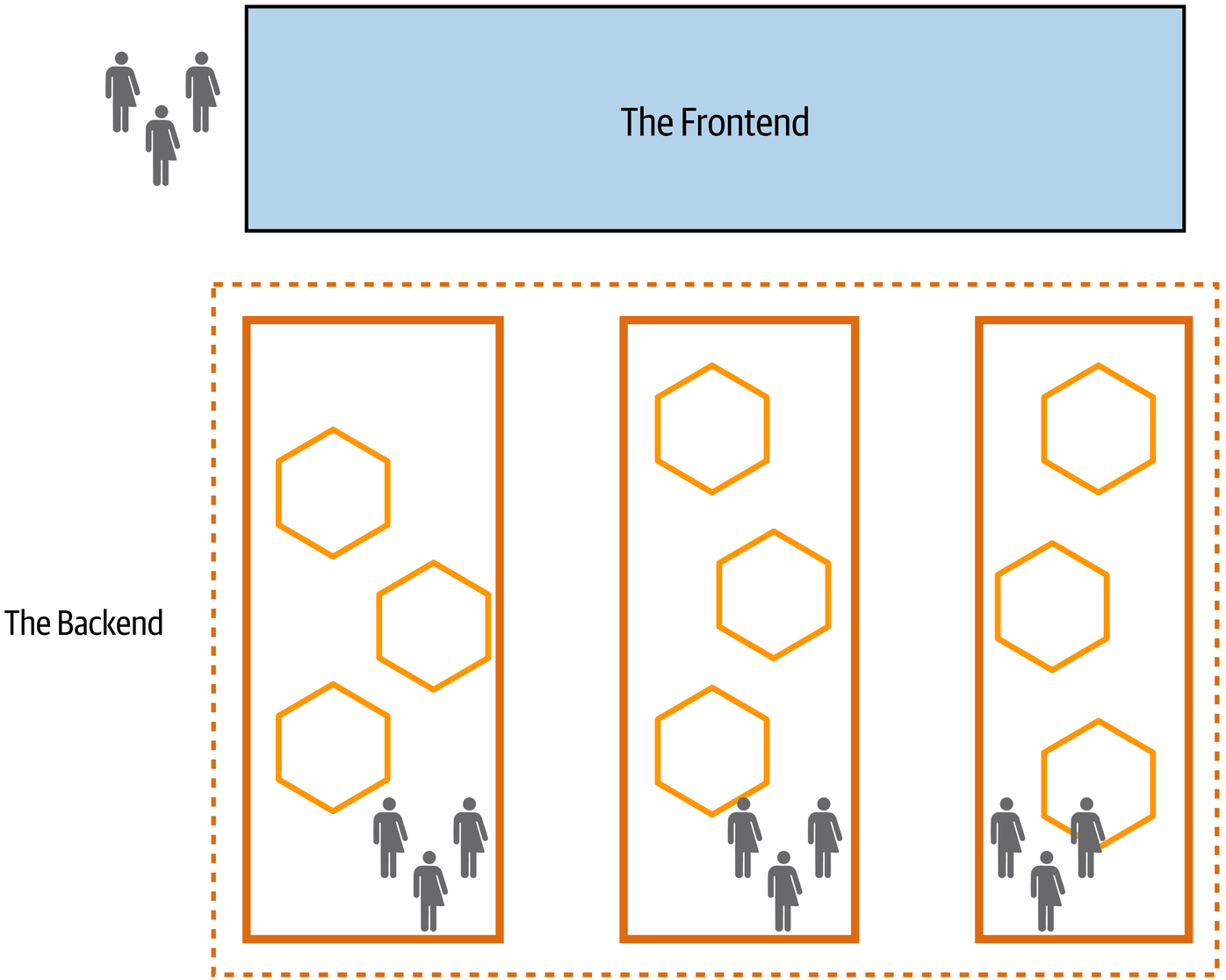


# THE UI?




**Still have lots of coordination**

# THE UI?




Still have lots of coordination

Need to push for teams to take ownership of the UI too

This means we need to break apart our frontends too!

# 2/4 Domain-Driven Design (in brief)

# POLL: WHAT'S YOUR EXPERIENCE LEVEL WITH DDD?

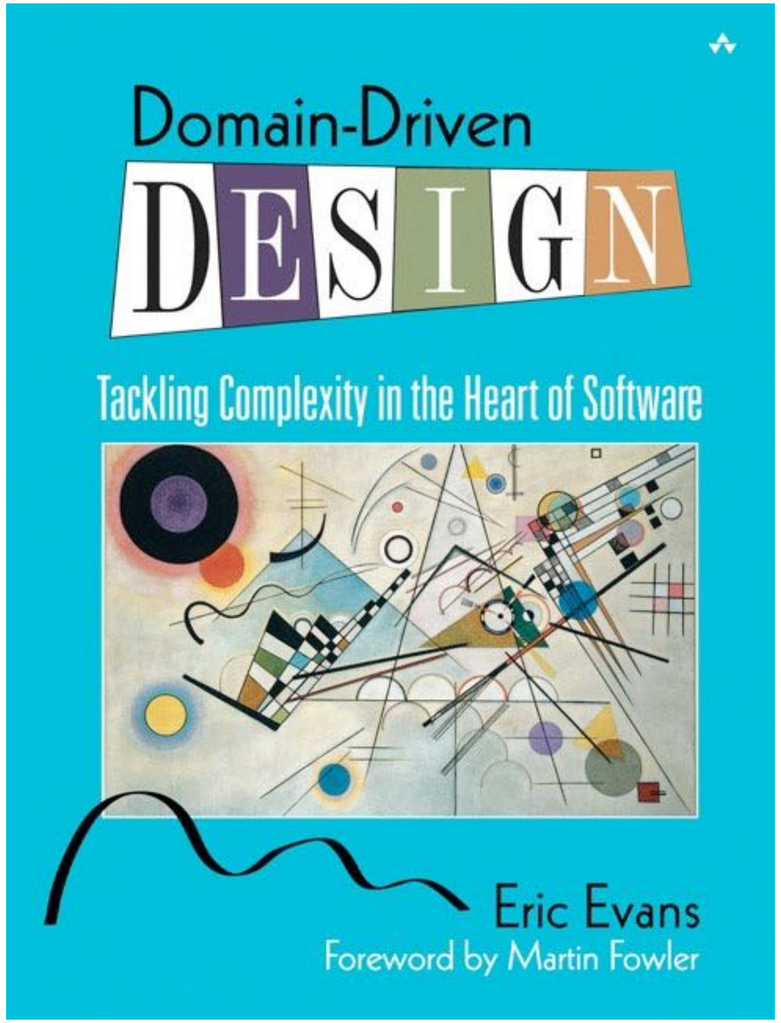
Never heard of it!

Heard of it, but not used it

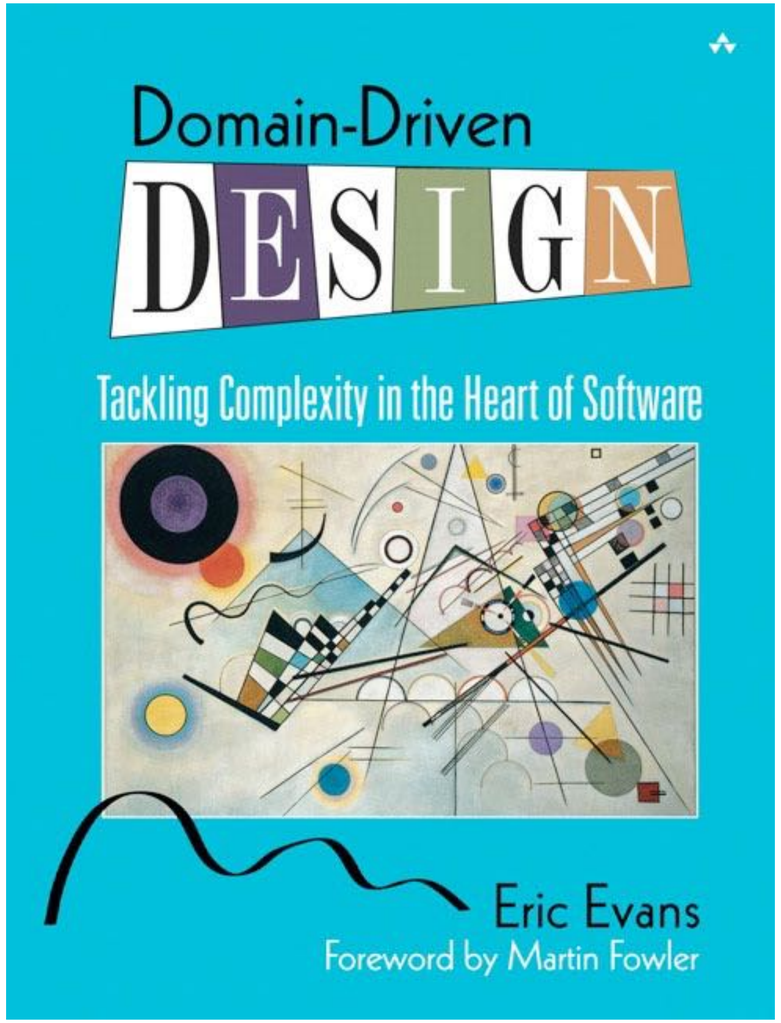
Tried using it a bit

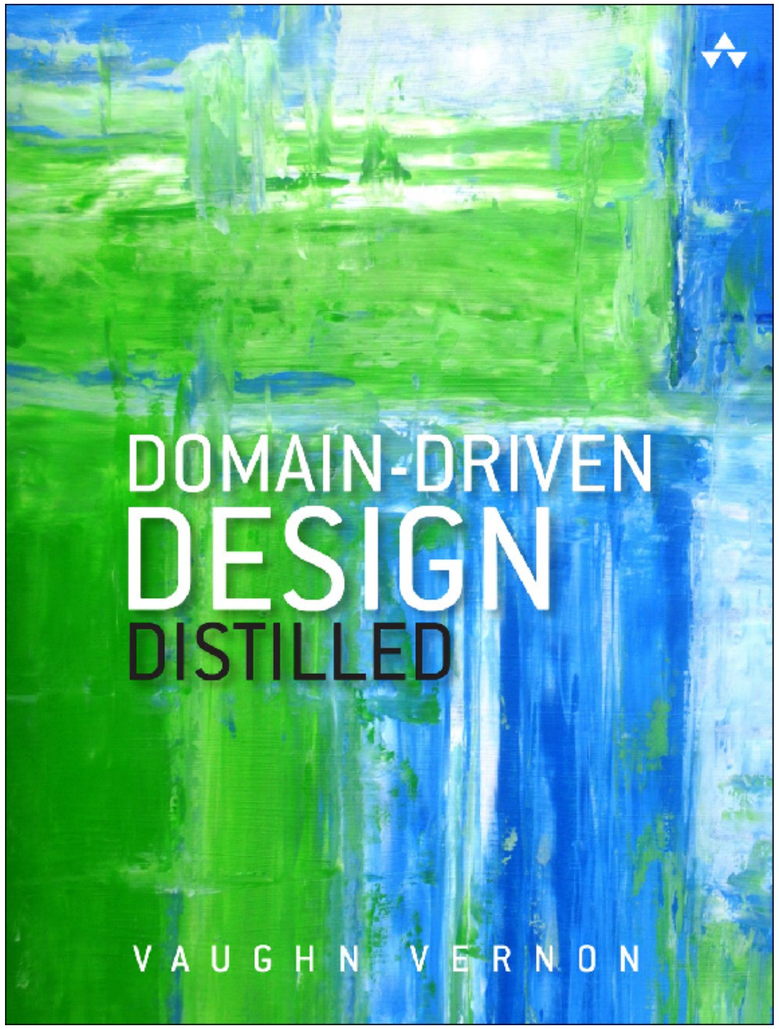
We use it all the time!

# DOMAIN DRIVEN DESIGN



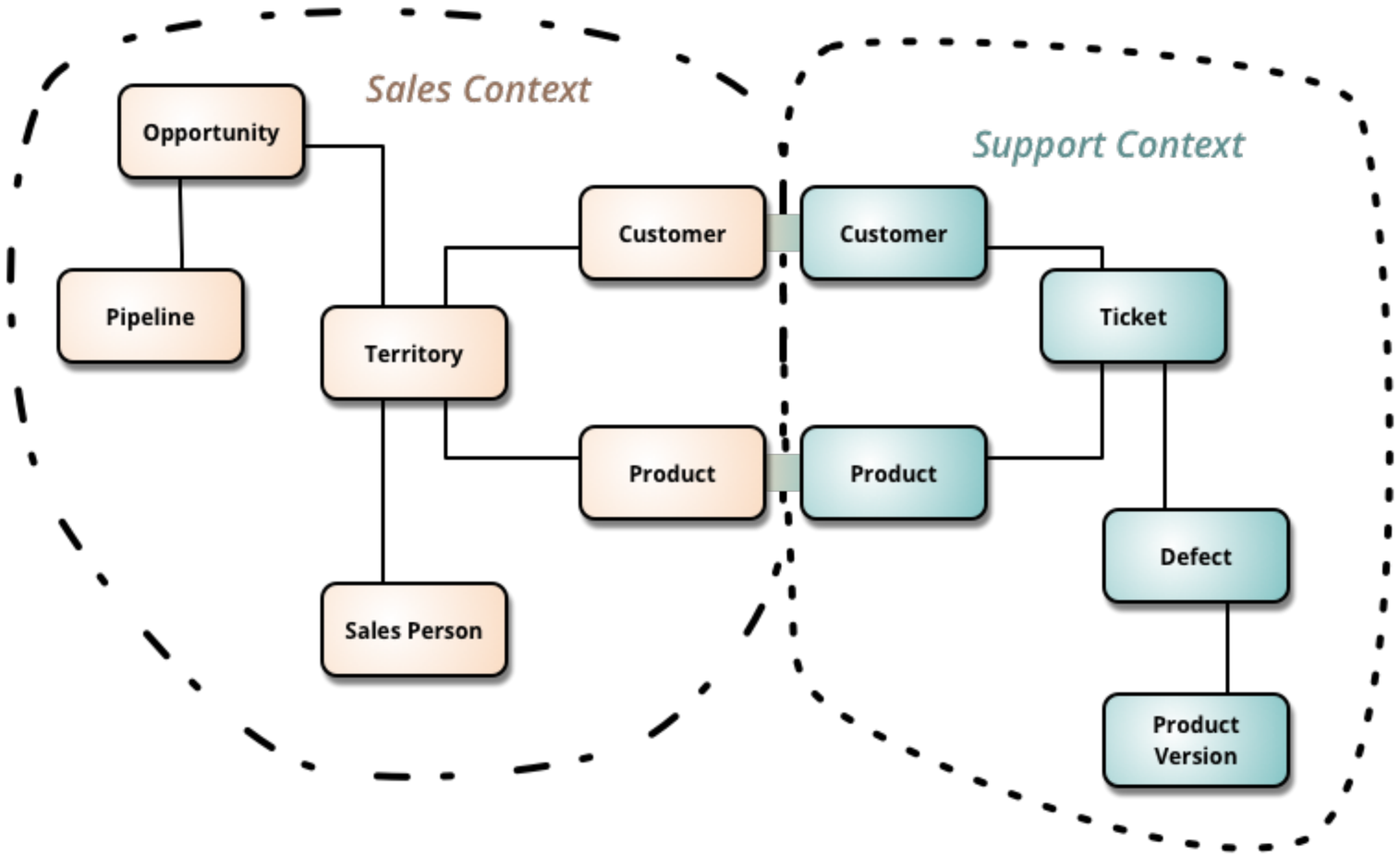
# DOMAIN DRIVEN DESIGN



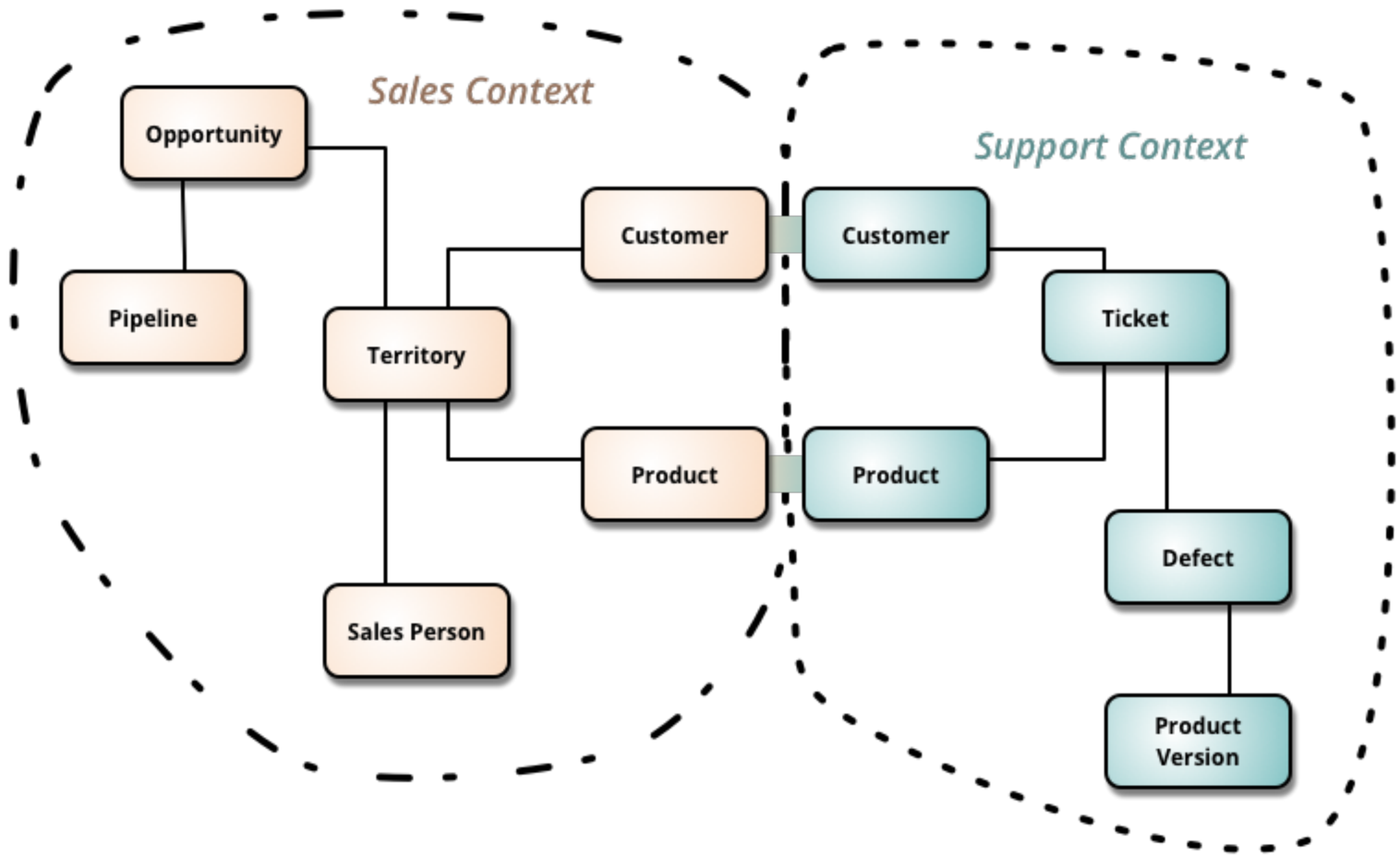
# Ubiquitous Language

# BOUNDED CONTEXTS




https://martinfowler.com/bliki/BoundedContext.html

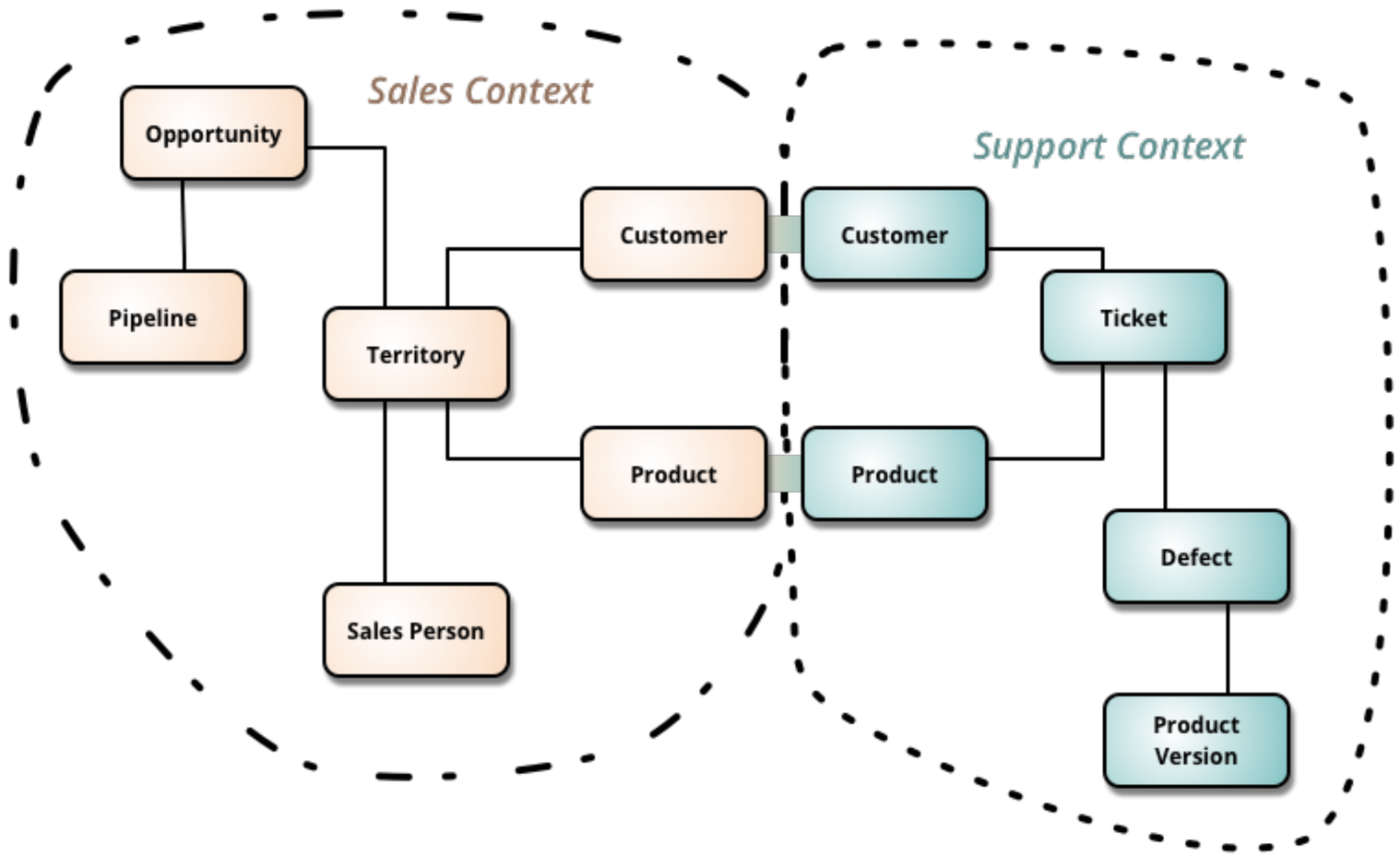
# BOUNDED CONTEXTS




Organisational groupings

https://martinfowler.com/bliki/BoundedContext.html

# BOUNDED CONTEXTS




Organisational groupings

Hiding complexity

https://martinfowler.com/bliki/BoundedContext.html

# AGGREGATES

## AGGREGATES

"A cluster of associated objects that are treated as a unit for the purpose of data changes"

*- Eric Evans, Domain-Driven Design*

## AGGREGATES

"A cluster of associated objects that are treated as a unit for the purpose of data changes"

*- Eric Evans, Domain-Driven Design*

We want to manage an aggregate as a single "entity" in terms of state changes

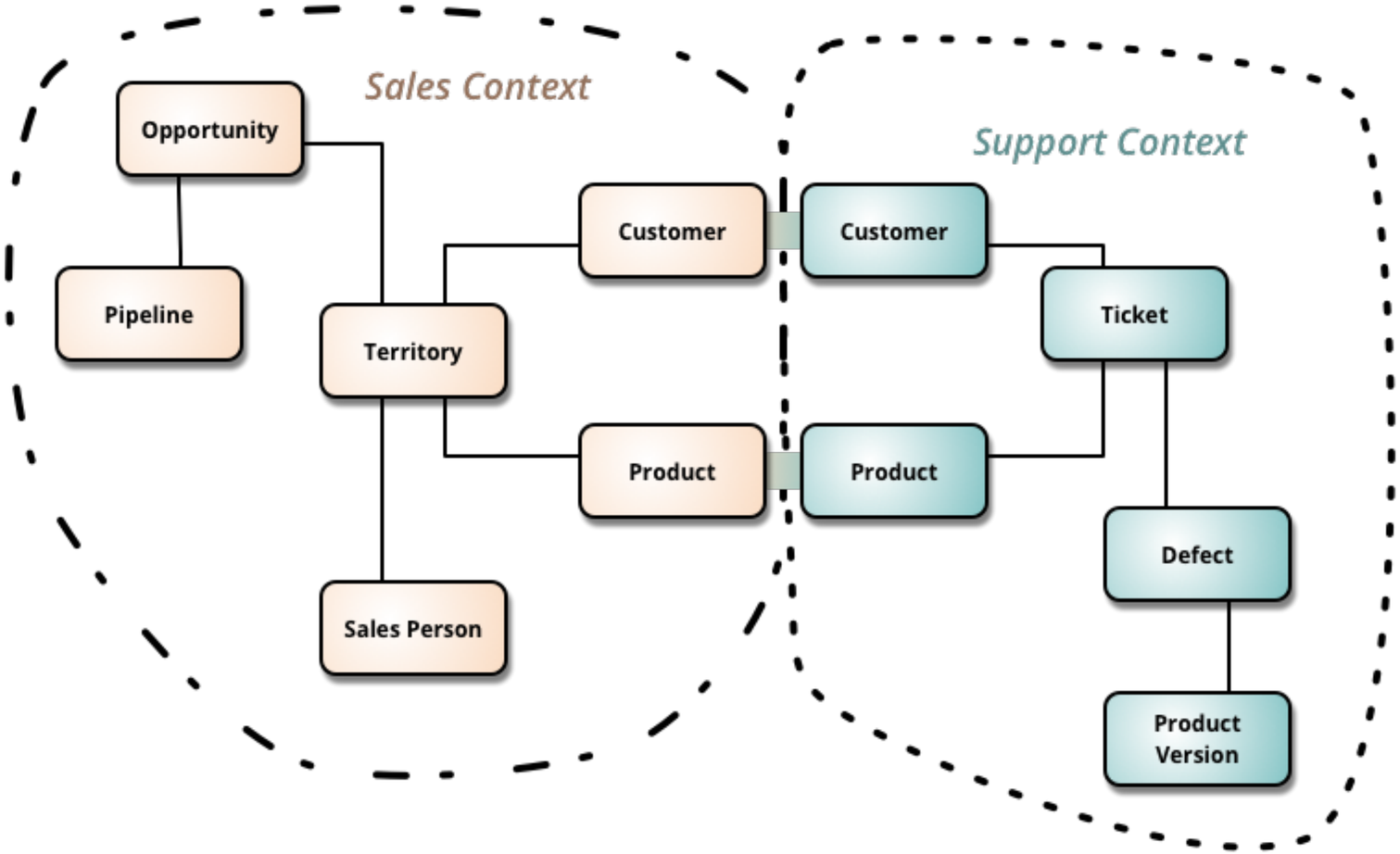
## AGGREGATES

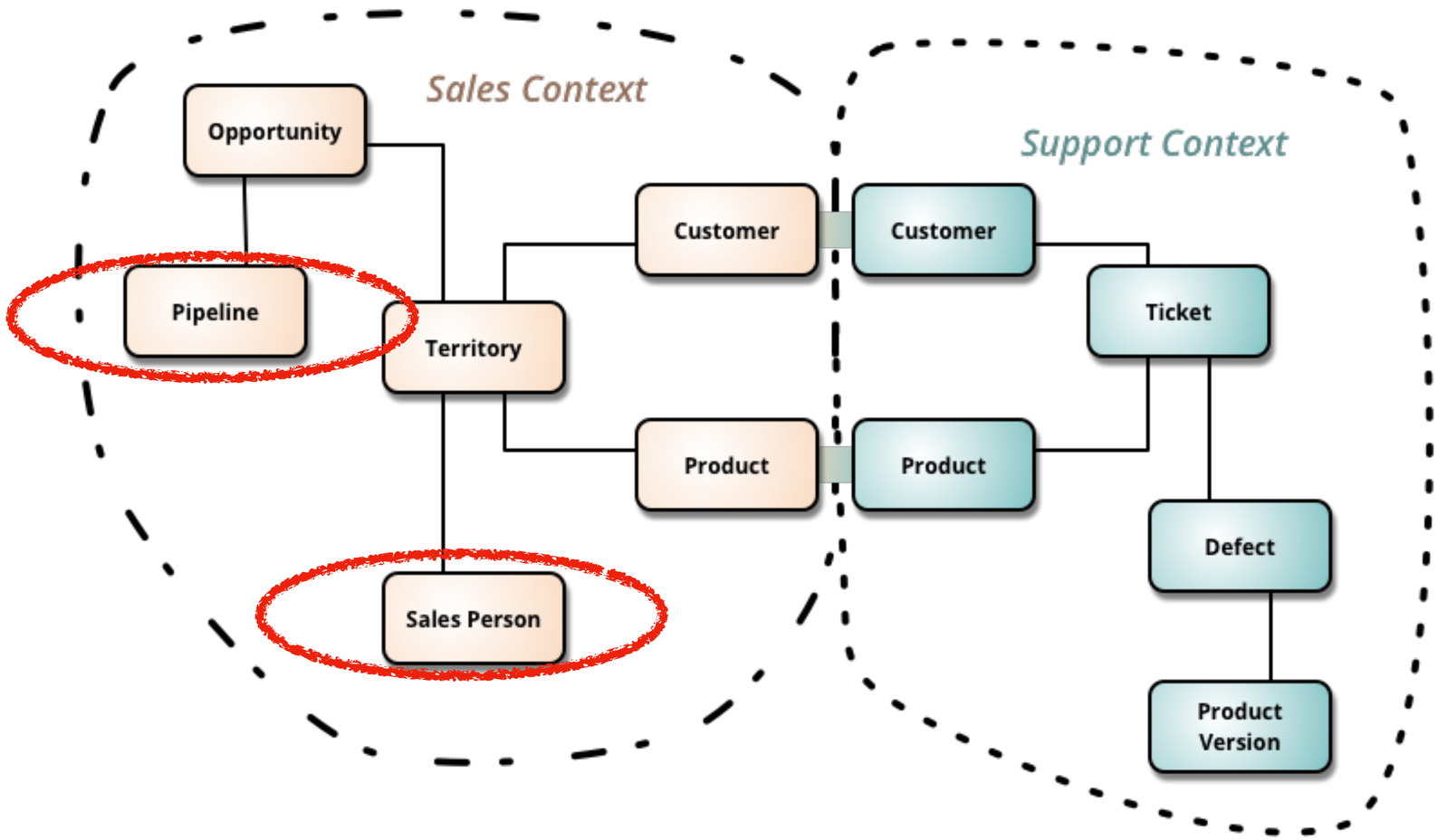
"A cluster of associated objects that are treated as a unit for the purpose of data changes"

*- Eric Evans, Domain-Driven Design*

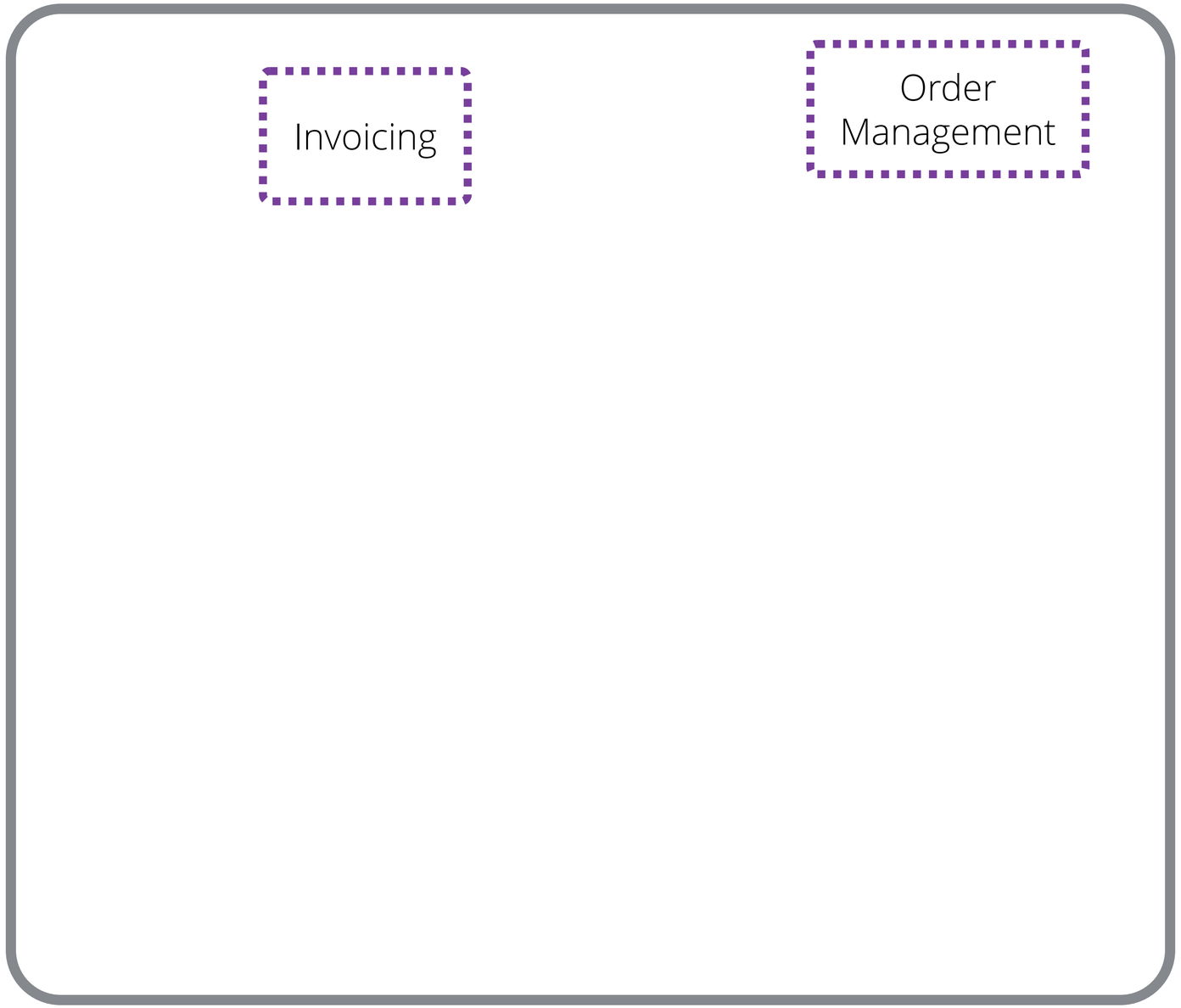
We want to manage an aggregate as a single "entity" in terms of state changes

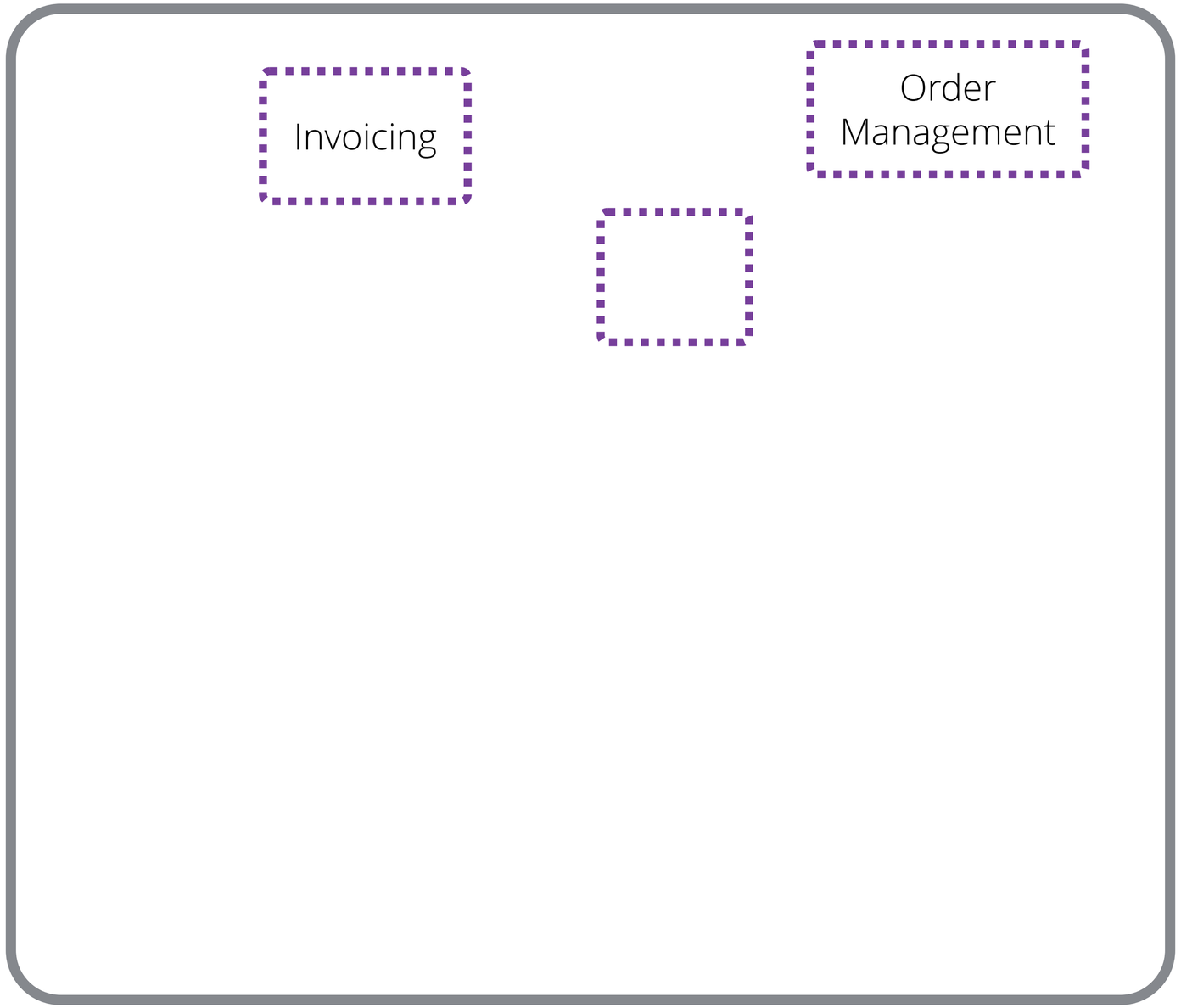
We want all operations which manage the state to behave in a consistent fashion

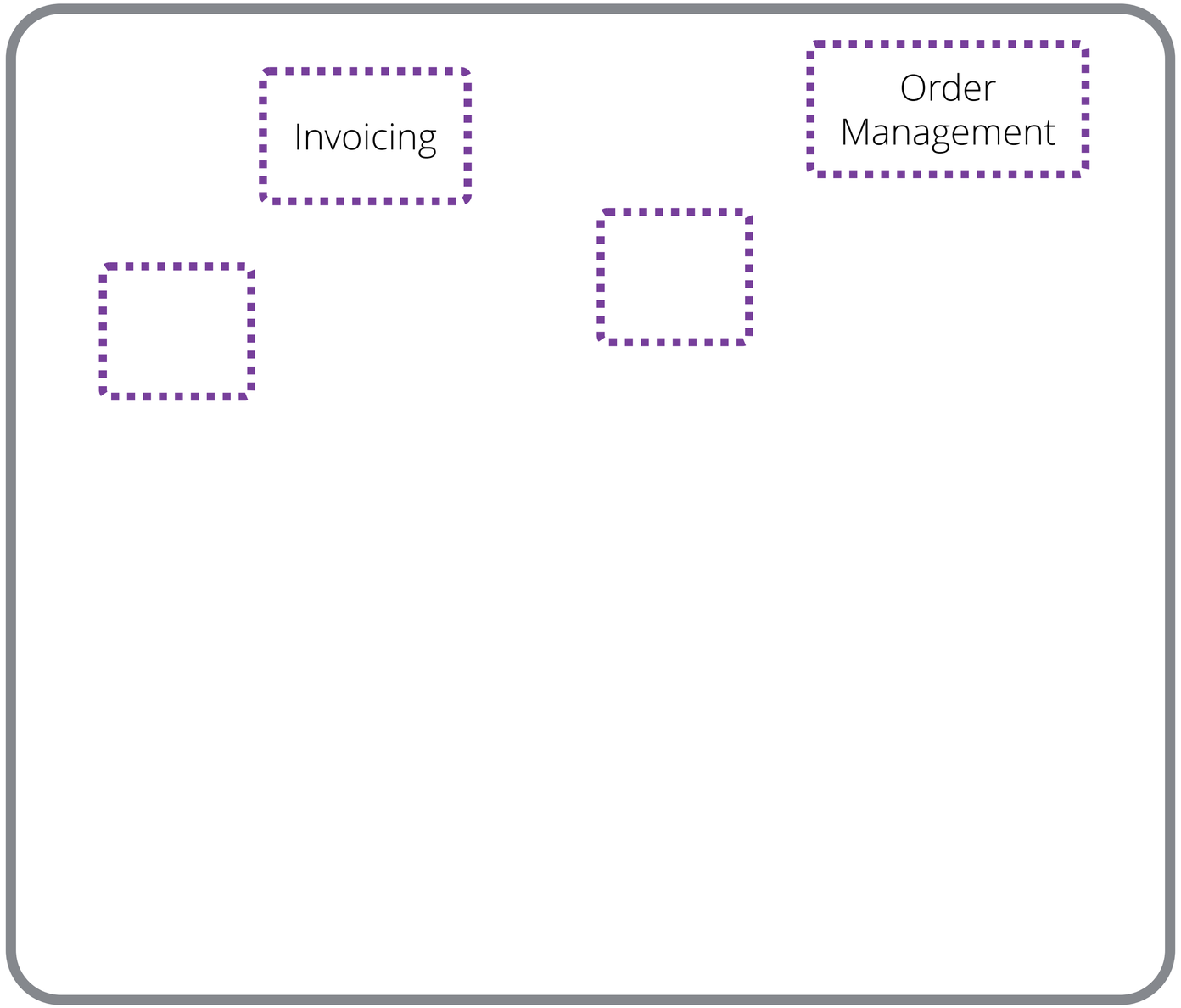

Sales Context
Support Context
Opportunity
Pipeline
Territory
Sales Person
Customer
Product
Customer
Product
Ticket
Defect
Product Version

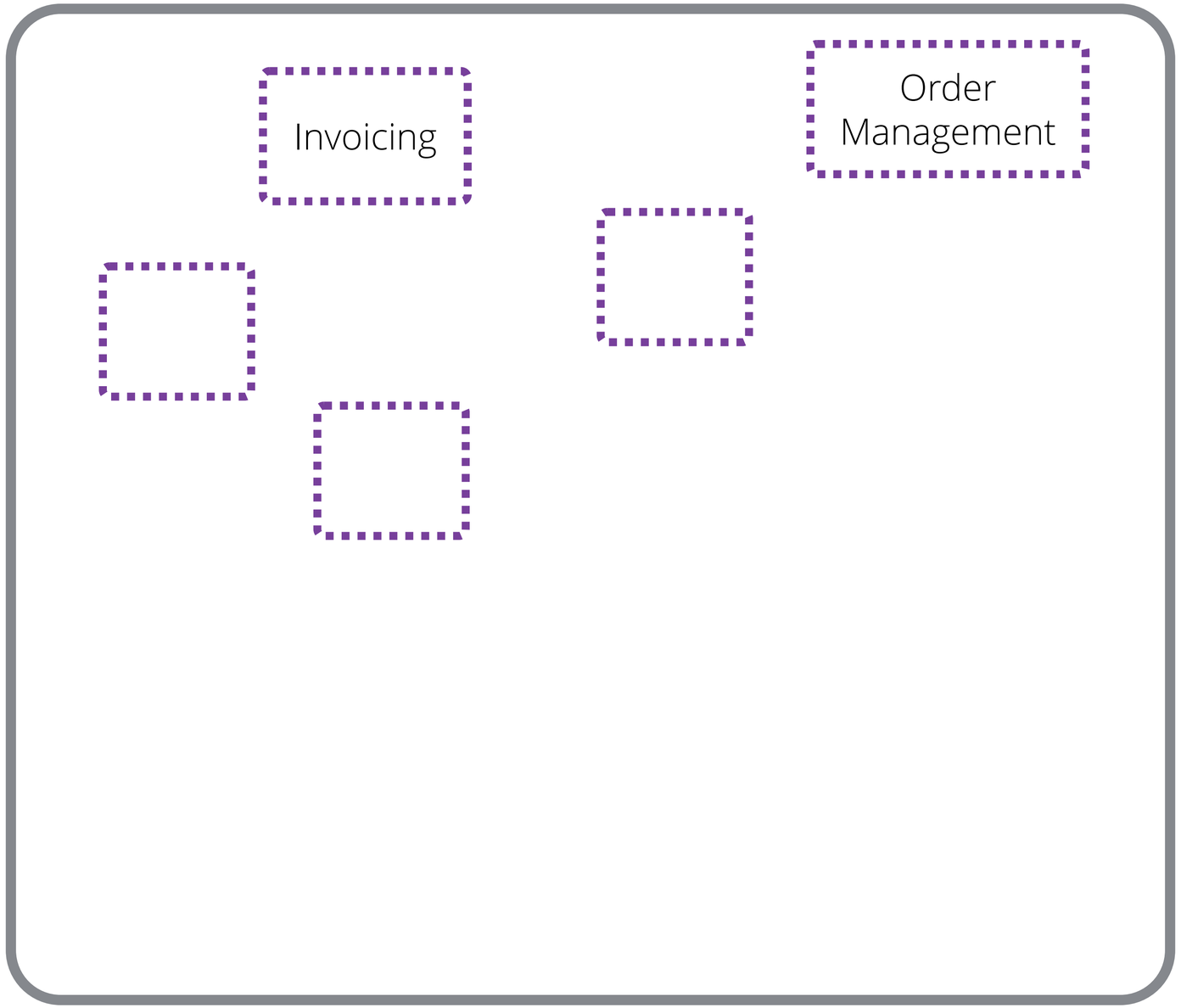

Sales Context
Support Context
Opportunity
Pipeline
Territory
Customer
Product
Sales Person
Customer
Ticket
Product
Defect
Product Version

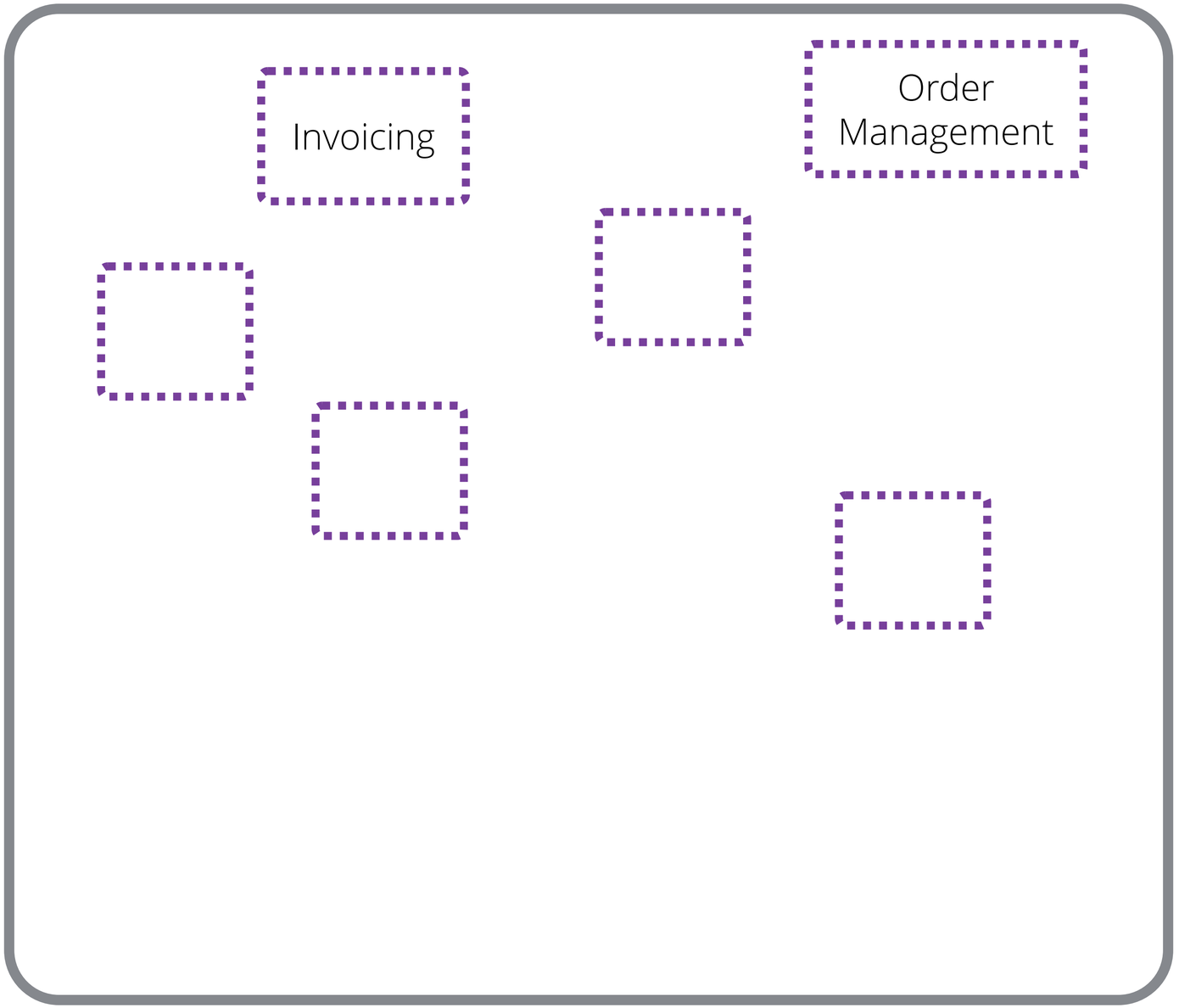

Order
Management

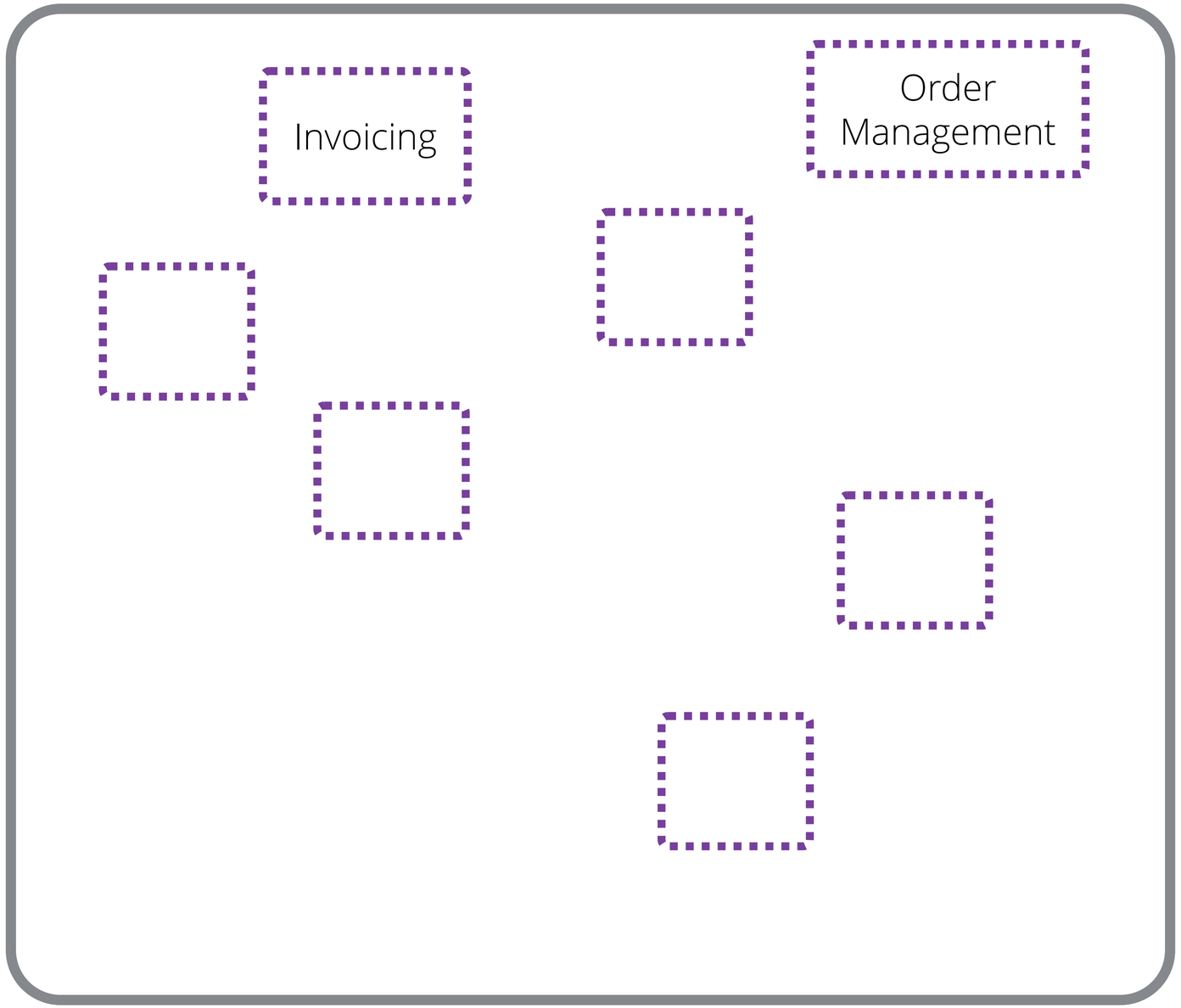

Invoicing
Order Management

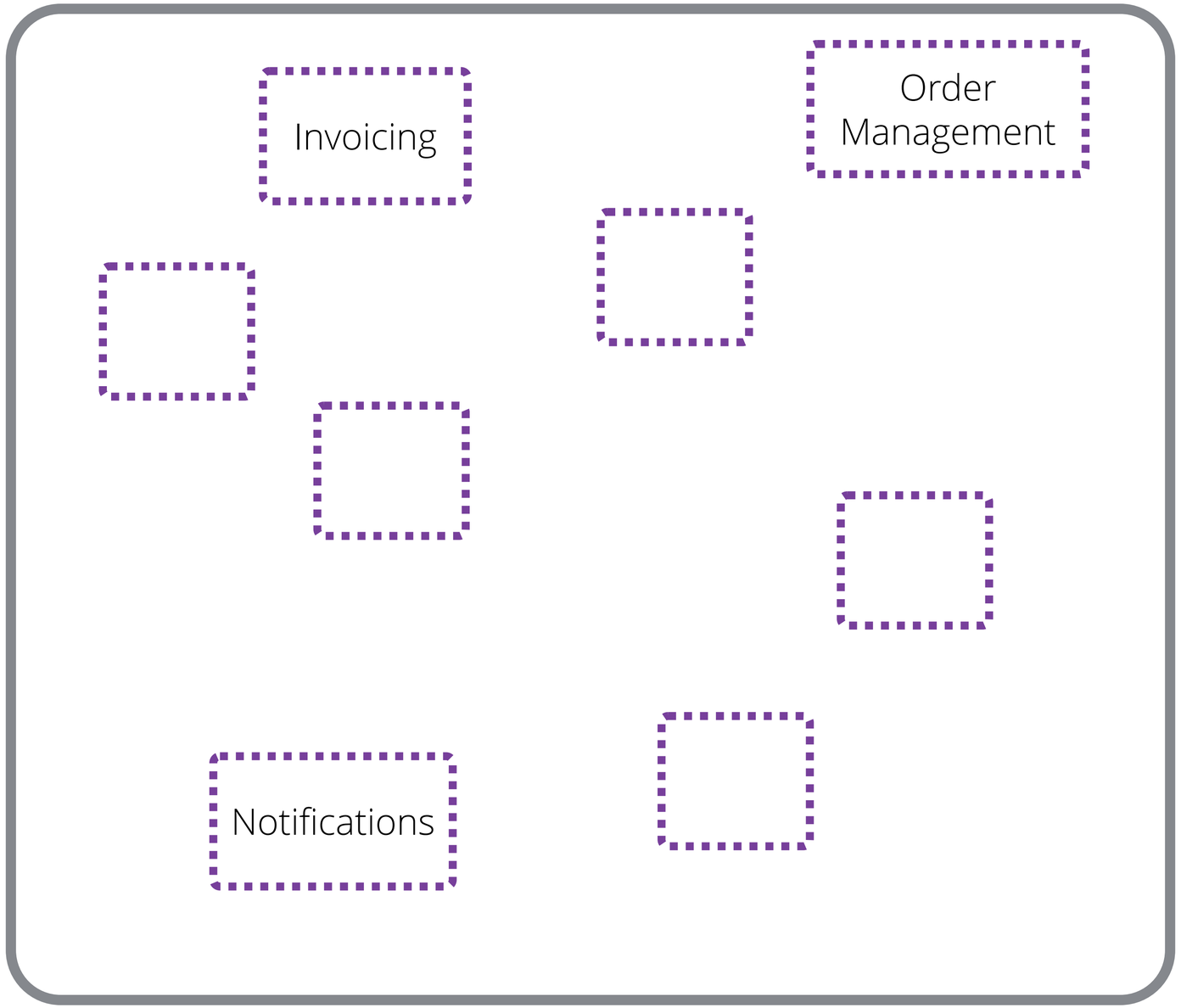

Invoicing
Order Management

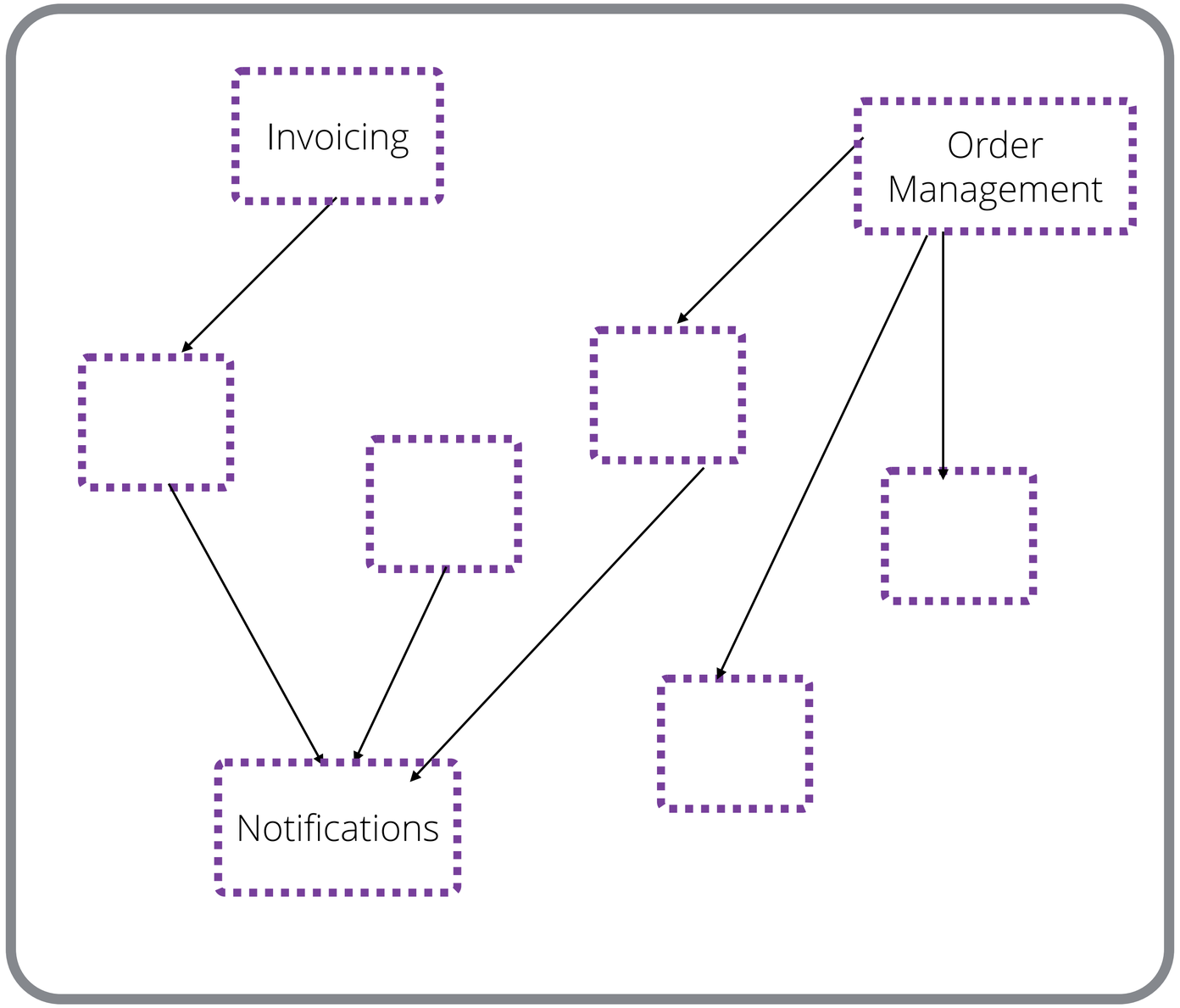

Invoicing
Order Management


Invoicing
Order Management

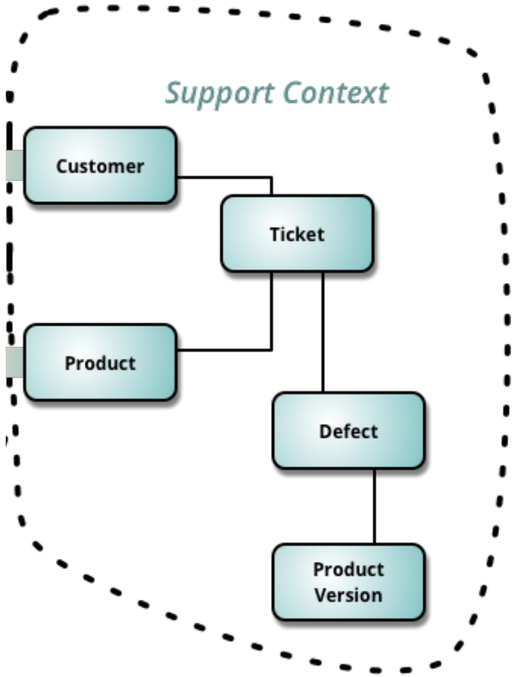

Invoicing
Order
Management


Invoicing
Order Management

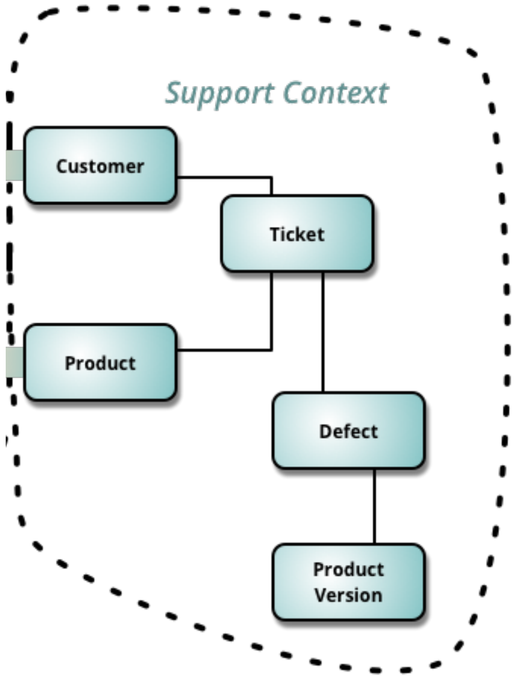

Invoicing
Order Management
Notifications

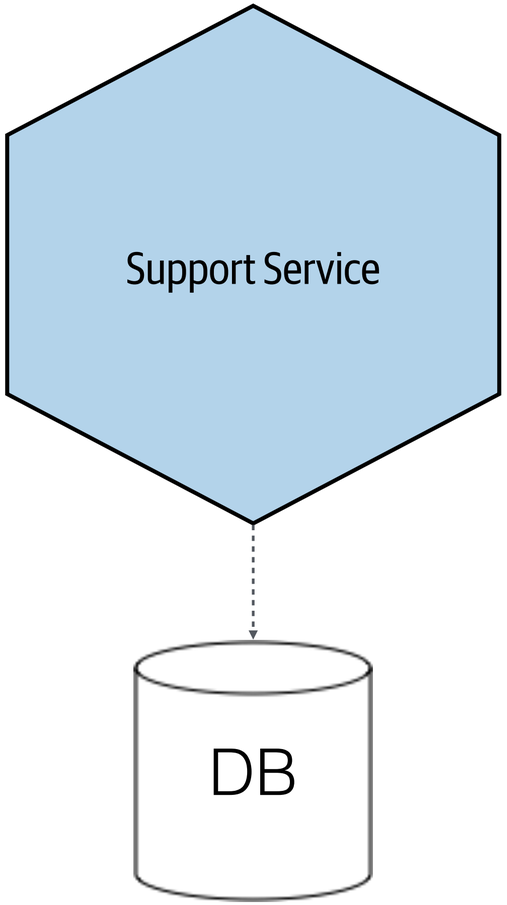

Invoicing
Order Management
Notifications

# IMPLEMENTATION EXAMPLE…

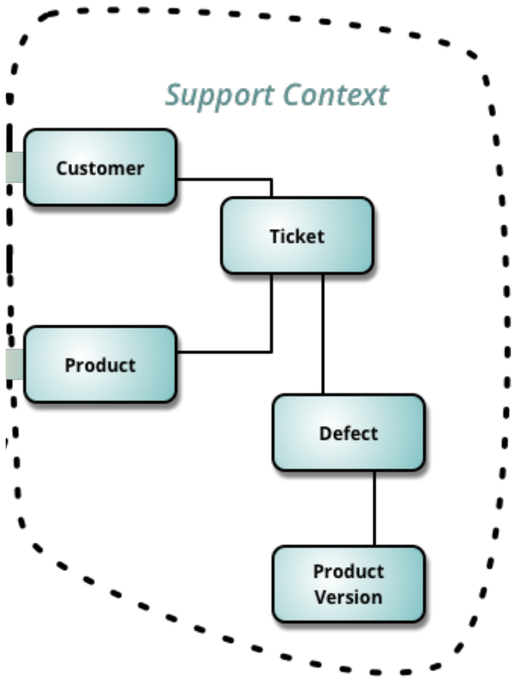


# IMPLEMENTATION EXAMPLE…

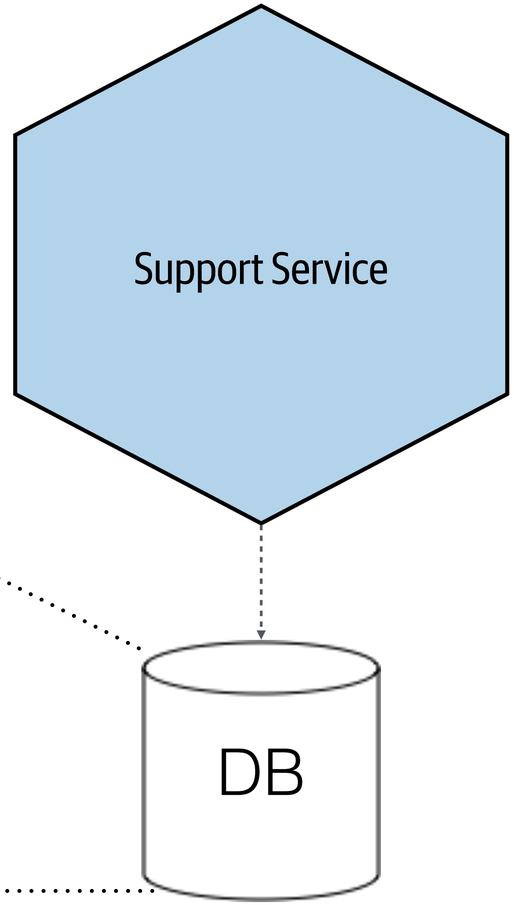


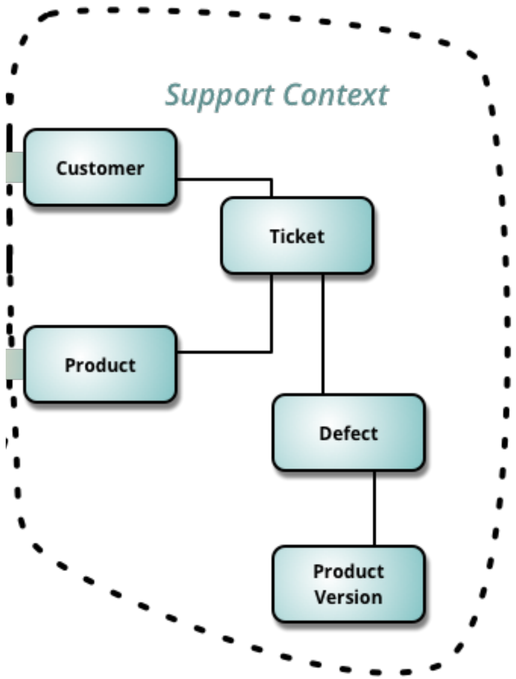
# IMPLEMENTATION EXAMPLE…

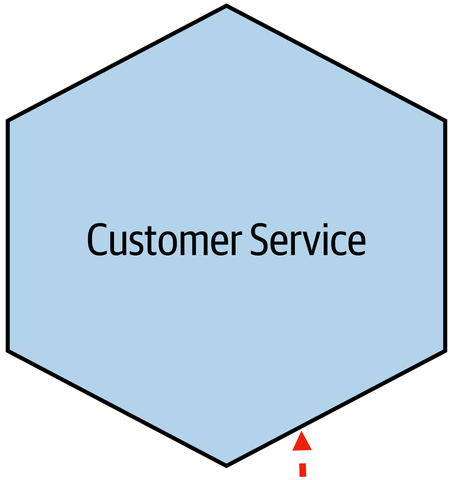


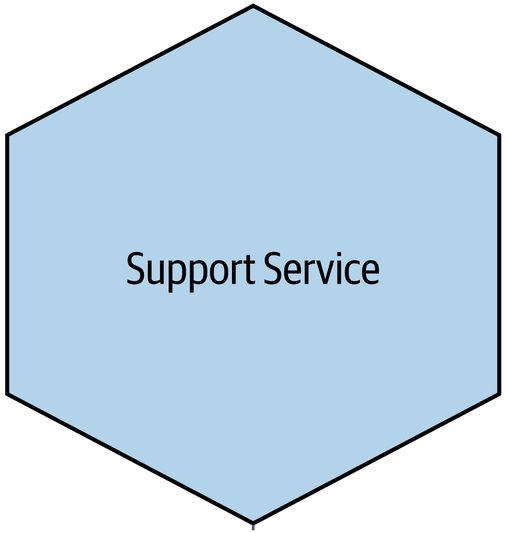

| Ticket ID | Customer ID | Subject | … |
|---|---|---|---|
| 123 | as45-244 | Software Doesn't Work! | |
| 456 | 88rhj-424 | Can't upgrade | |



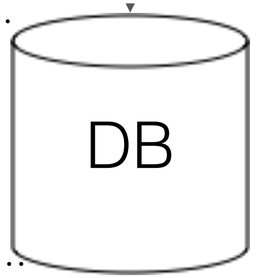
# IMPLEMENTATION EXAMPLE…



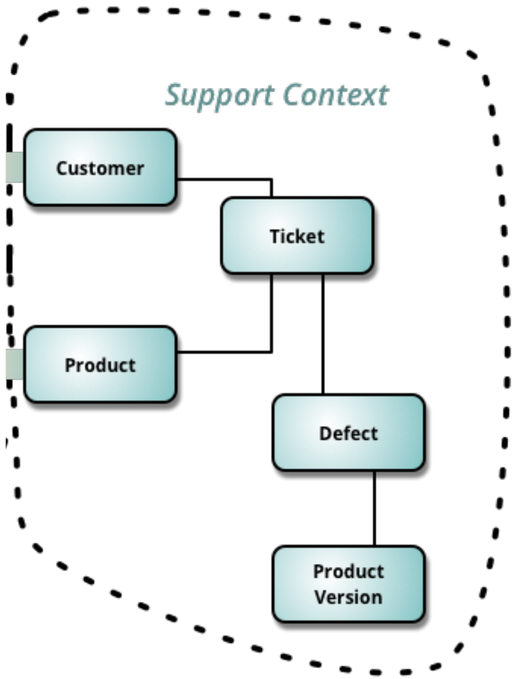



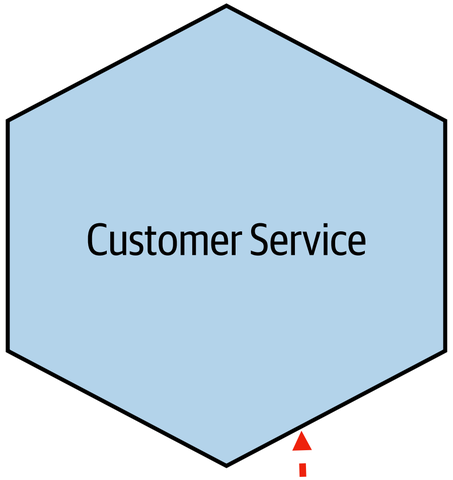




| Ticket ID | Customer ID | Subject | … |
|---|---|---|---|
| 123 | as45-244 | Software Doesn't Work! | |
| 456 | 88rhj-424 | Can't upgrade | |

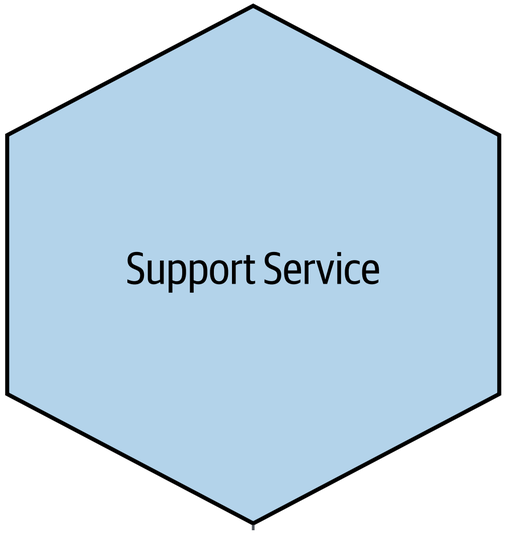


# IMPLEMENTATION EXAMPLE…

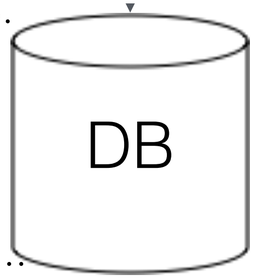


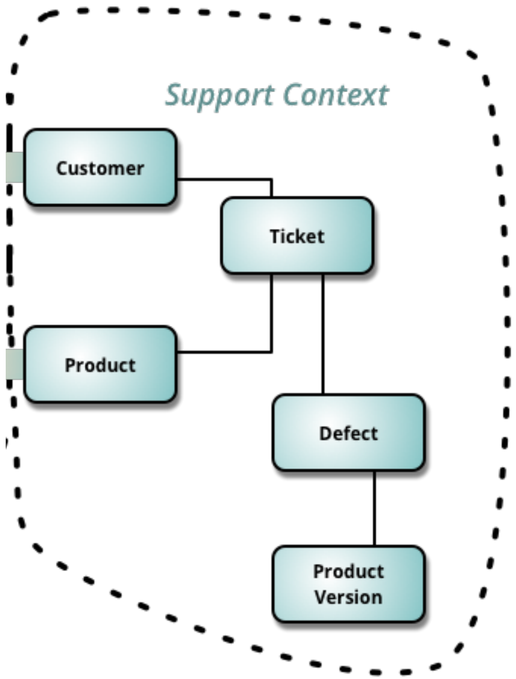

**Referencing a “foreign” aggregate**

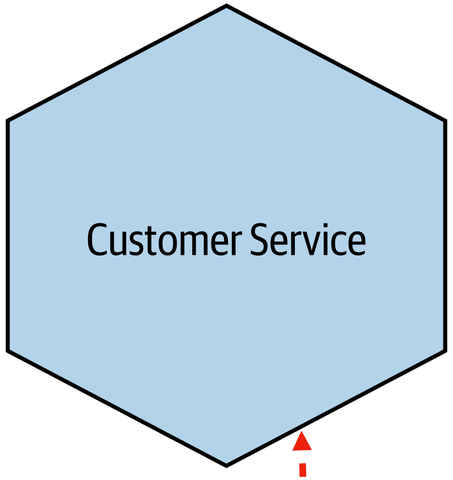



| Ticket ID | Customer ID | Subject | … |
|---|---|---|---|
| 123 | as45-244 | Software Doesn’t Work! | |
| 456 | 88rhj-424 | Can’t upgrade | |

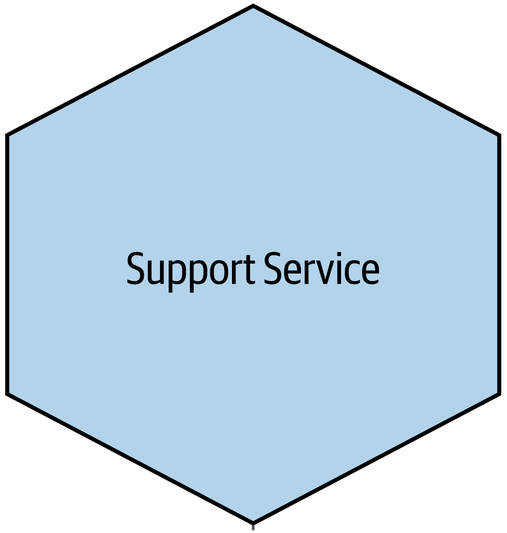


# IMPLEMENTATION EXAMPLE…



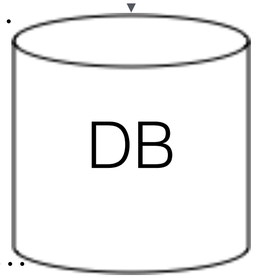




**Referencing a “foreign” aggregate**




| Ticket ID | Customer ID | Subject | … |
|---|---|---|---|
| 123 | as45-244 | Software Doesn’t Work! | |
| 456 | 88rhj-424 | Can’t upgrade | |

**Aggregate**



# PSEUDO-URIS FOR CROSS-MICROSERVICE REFERENCES

## Pattern: Using Pseudo-URIs with Microservices

Mar 22, 2017

• Microservices • Distributed Systems • Patterns •

I've spent some time talking about the very basics you need to have in place before thinking about going down a microservices route, but even if you have these in place that doesn't mean that you aren't going to find some new surprises. Microservices impose a very distributed architecture, and this requires us to re-visit many concepts that are considered a solved problem in more traditional scenarios.

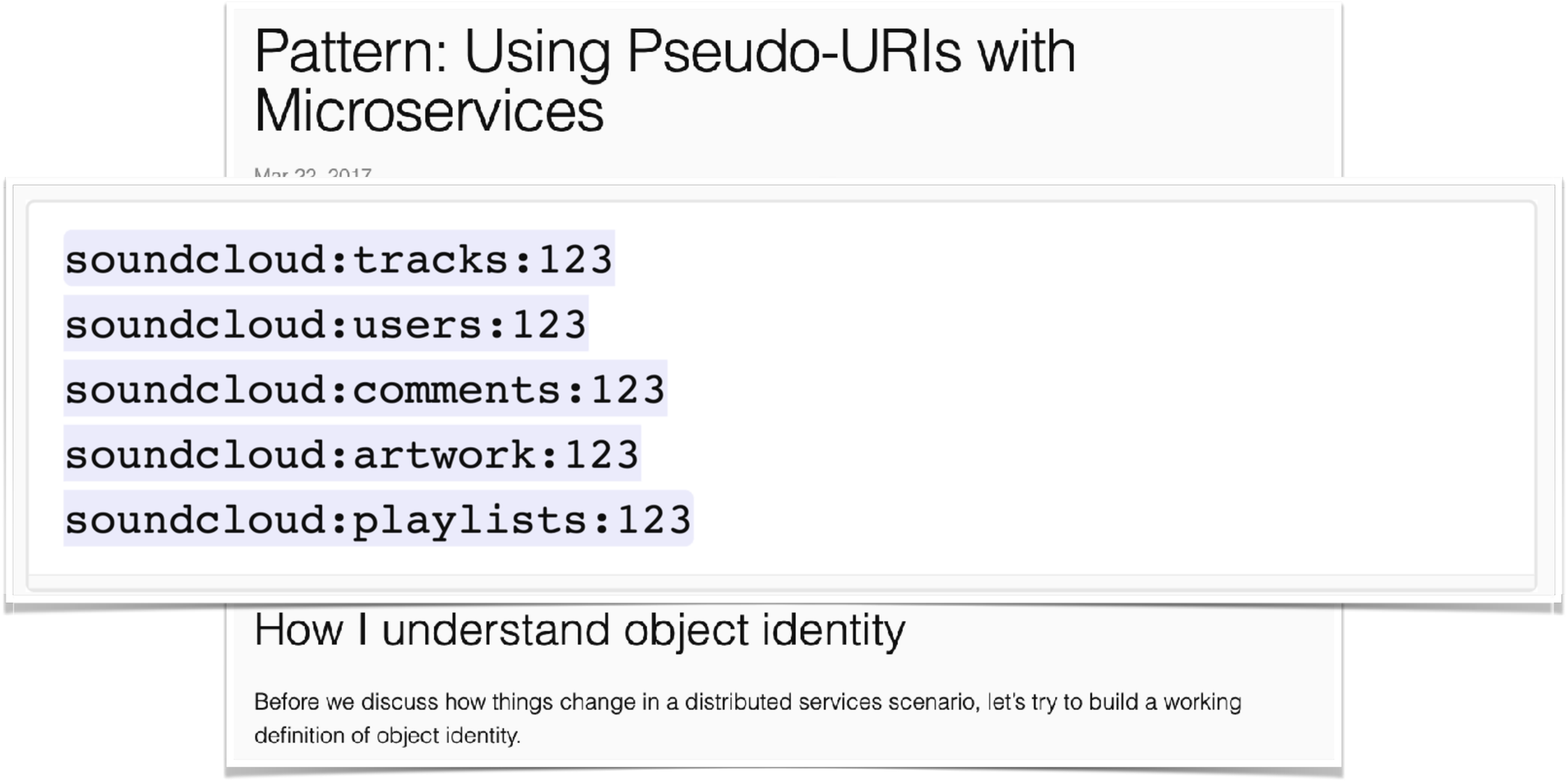
One of such concepts is how we implement identities for objects. This is usually a no-brainer in more monolithic architectures but it becomes a more interesting challenge as we have more distribution and collaboration between services.

### How I understand object identity

Before we discuss how things change in a distributed services scenario, let's try to build a working definition of object identity.

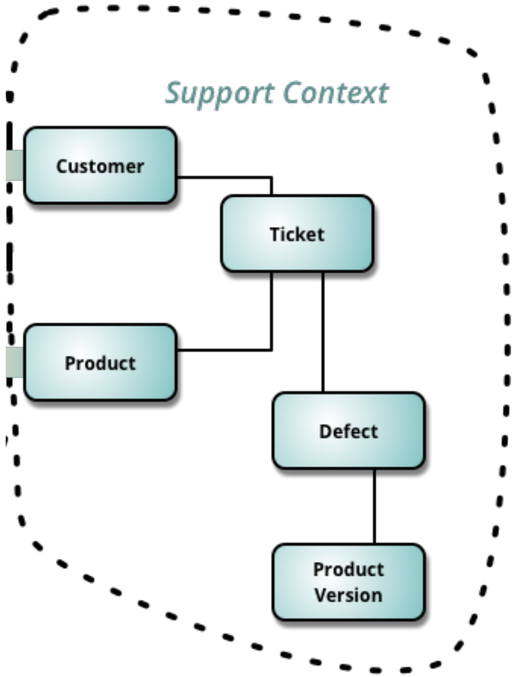
**https://philcalcado.com/2017/03/22/pattern_using_seudo-uris_with_microservices.html**

# PSEUDO-URIS FOR CROSS-MICROSERVICE REFERENCES

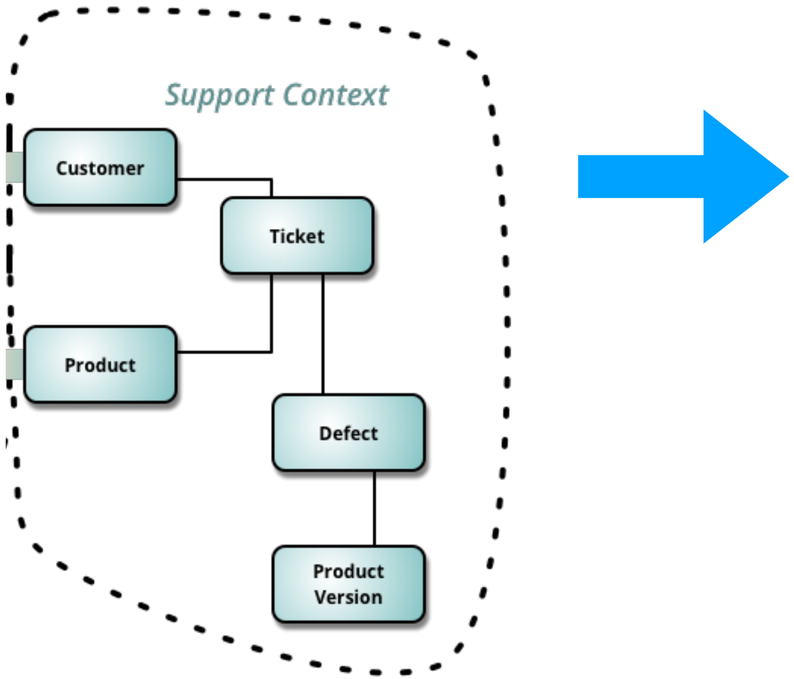



https://philcalcado.com/2017/03/22/pattern_using_seudo-uris_with_microservices.html

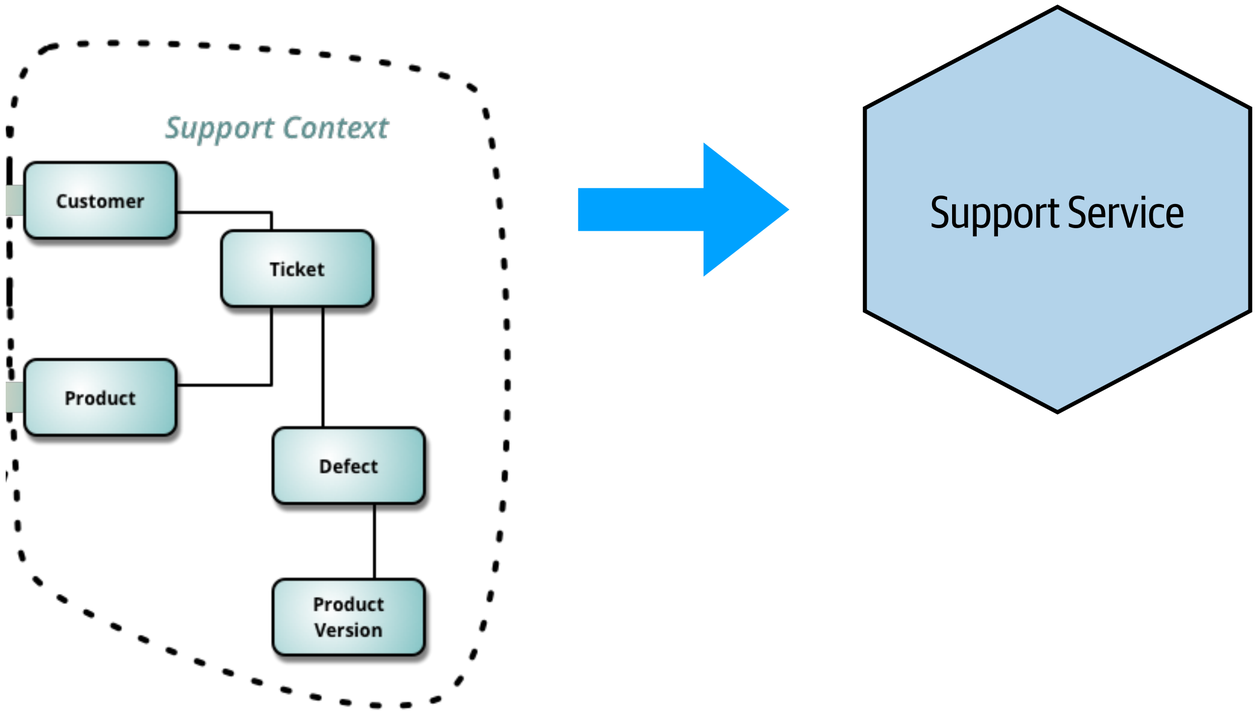
# MAPPING TO MICROSERVICES?



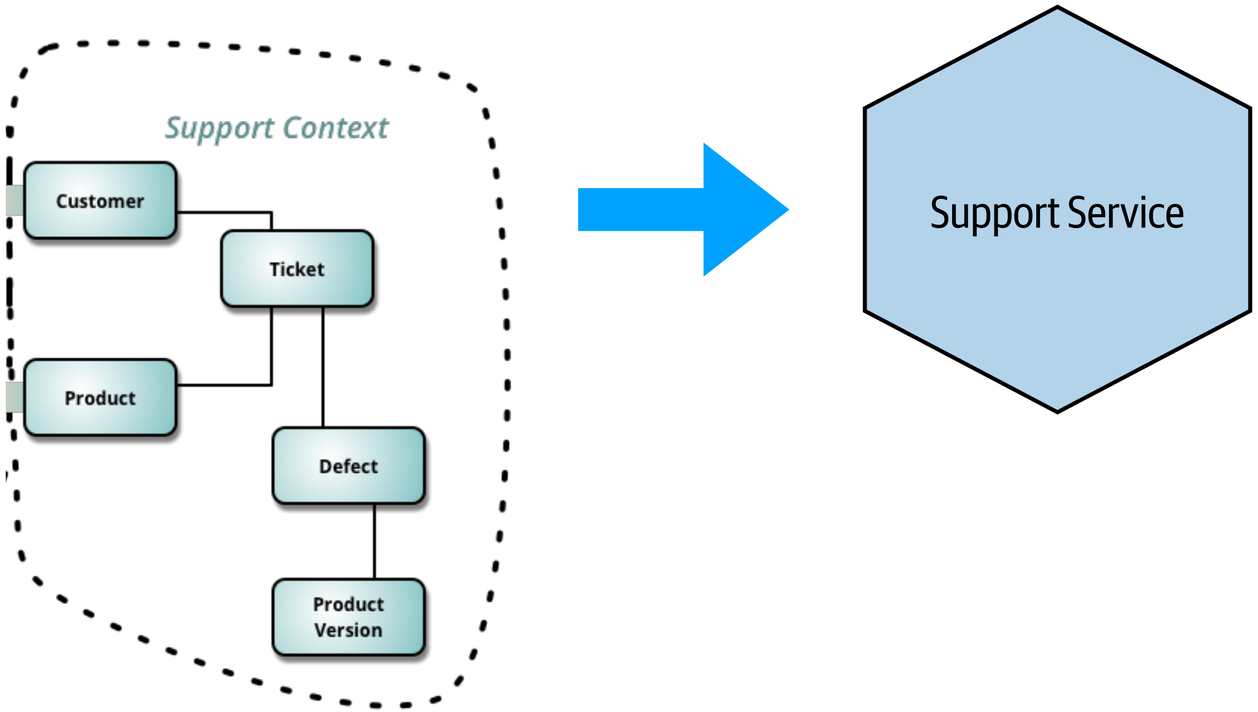
# MAPPING TO MICROSERVICES?



# MAPPING TO MICROSERVICES?

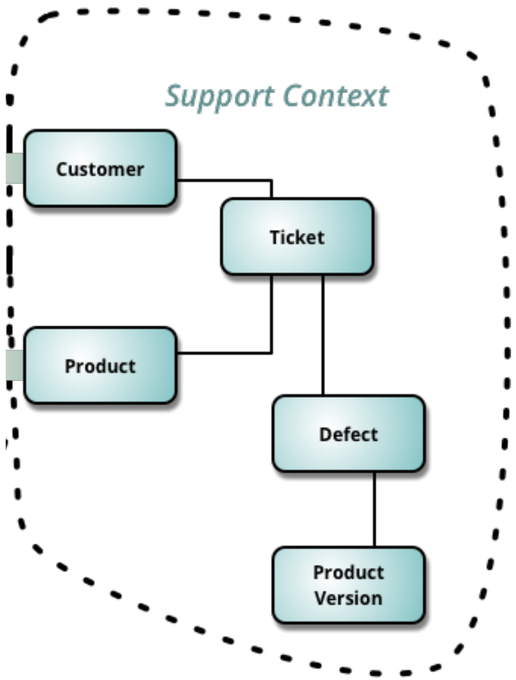


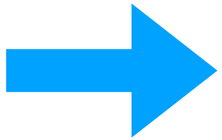
# MAPPING TO MICROSERVICES?

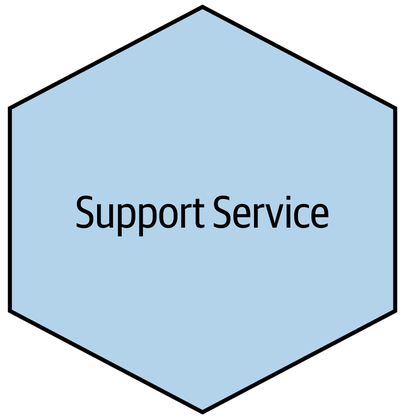



**Mapping bounded contexts to microservices is a good starting point**

# MAPPING TO MICROSERVICES?






Mapping bounded contexts to microservices is a good starting point

As you get more confident you might get more fine-grained…

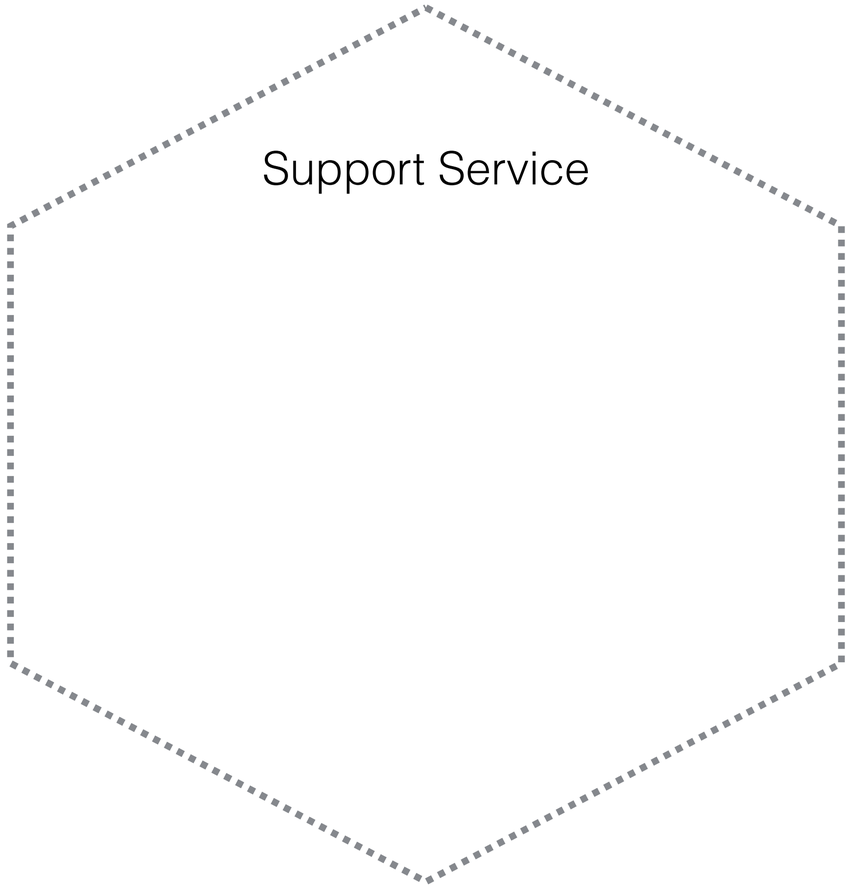
# GETTING FINER-GRAINED

## GETTING FINER-GRAINED

Can break down around aggregate boundaries

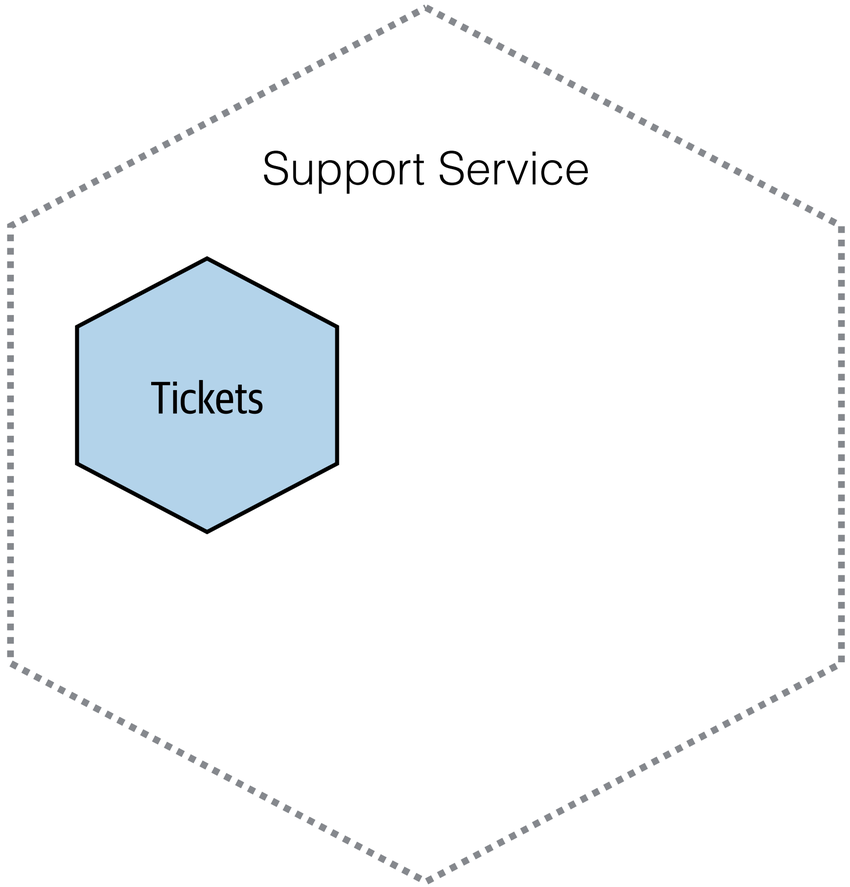
## GETTING FINER-GRAINED

**Can break down around aggregate boundaries**



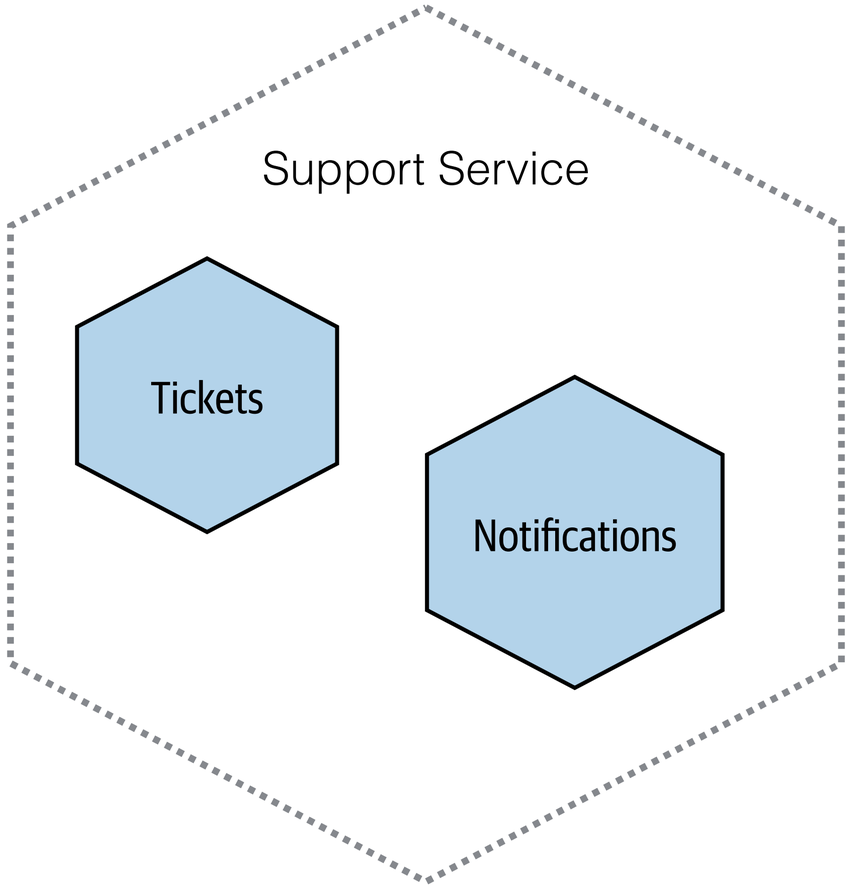
# GETTING FINER-GRAINED

**Can break down around aggregate boundaries**



## GETTING FINER-GRAINED

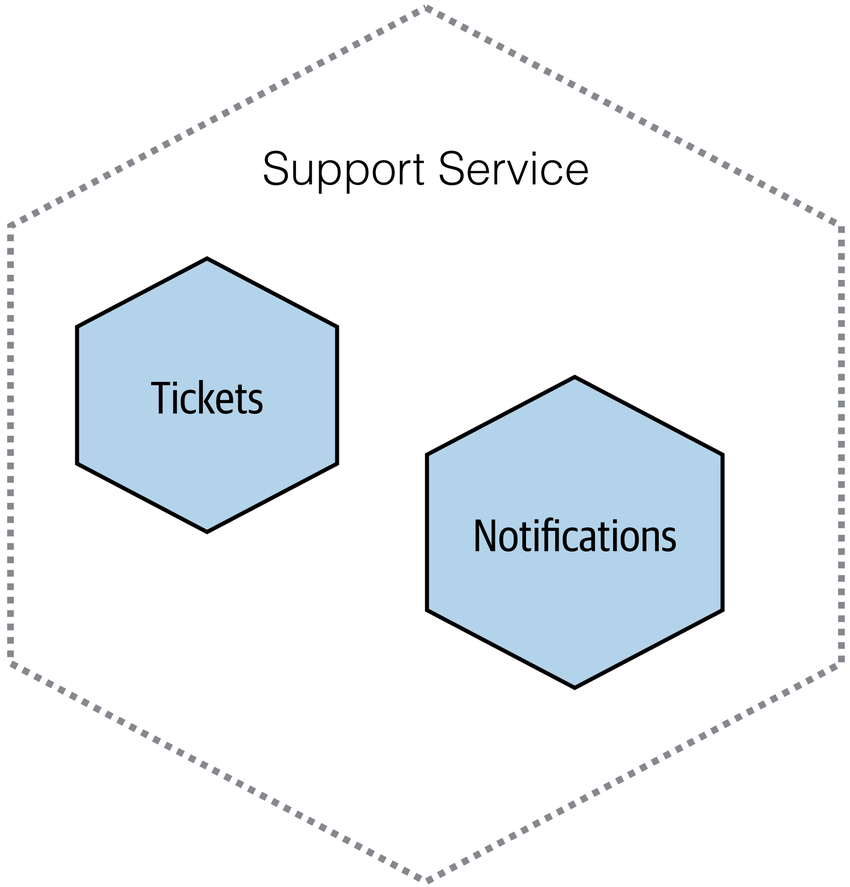
**Can break down around aggregate boundaries**



# GETTING FINER-GRAINED

**Can break down around aggregate boundaries**

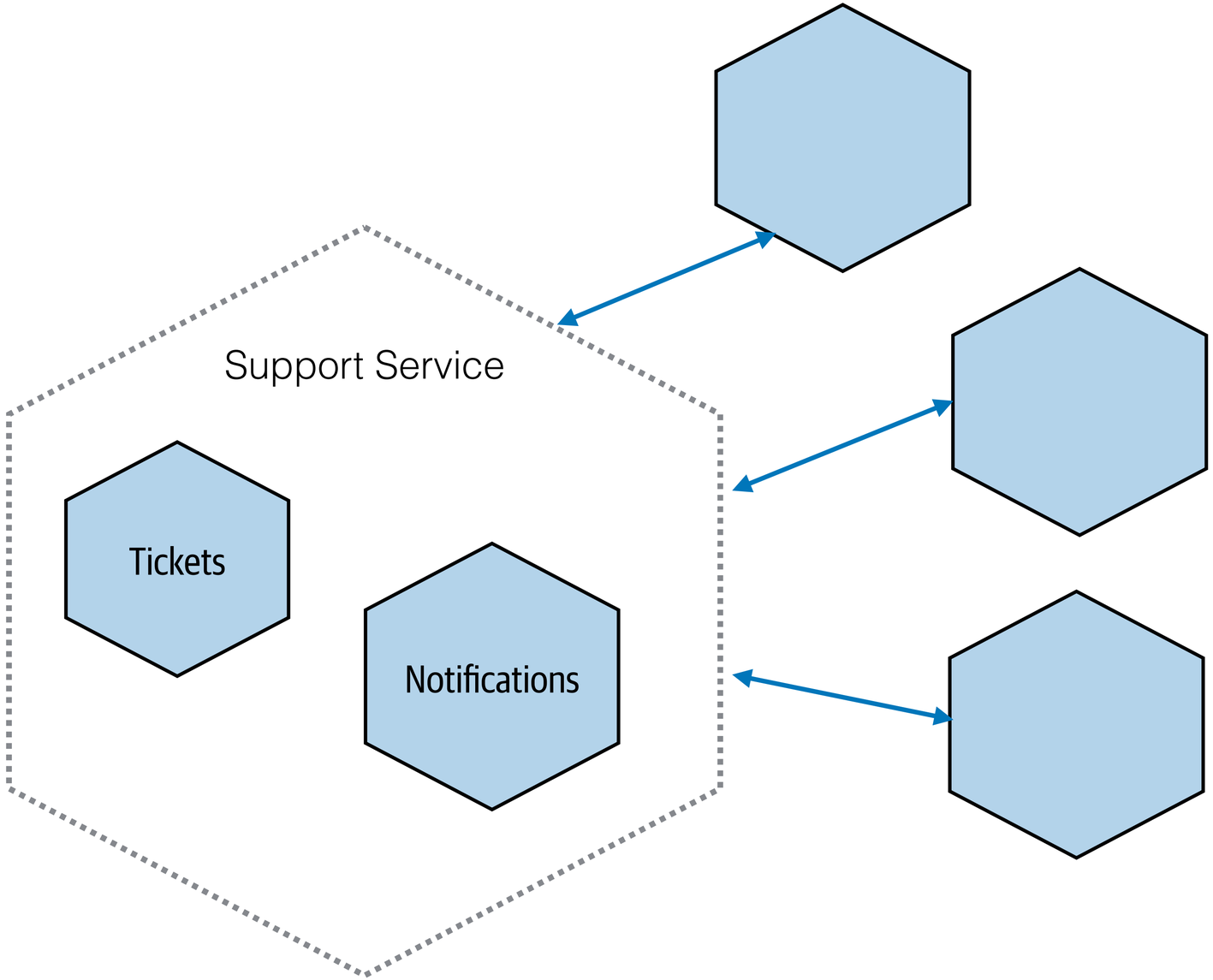
**Keep the higher-level abstraction for outside parties**



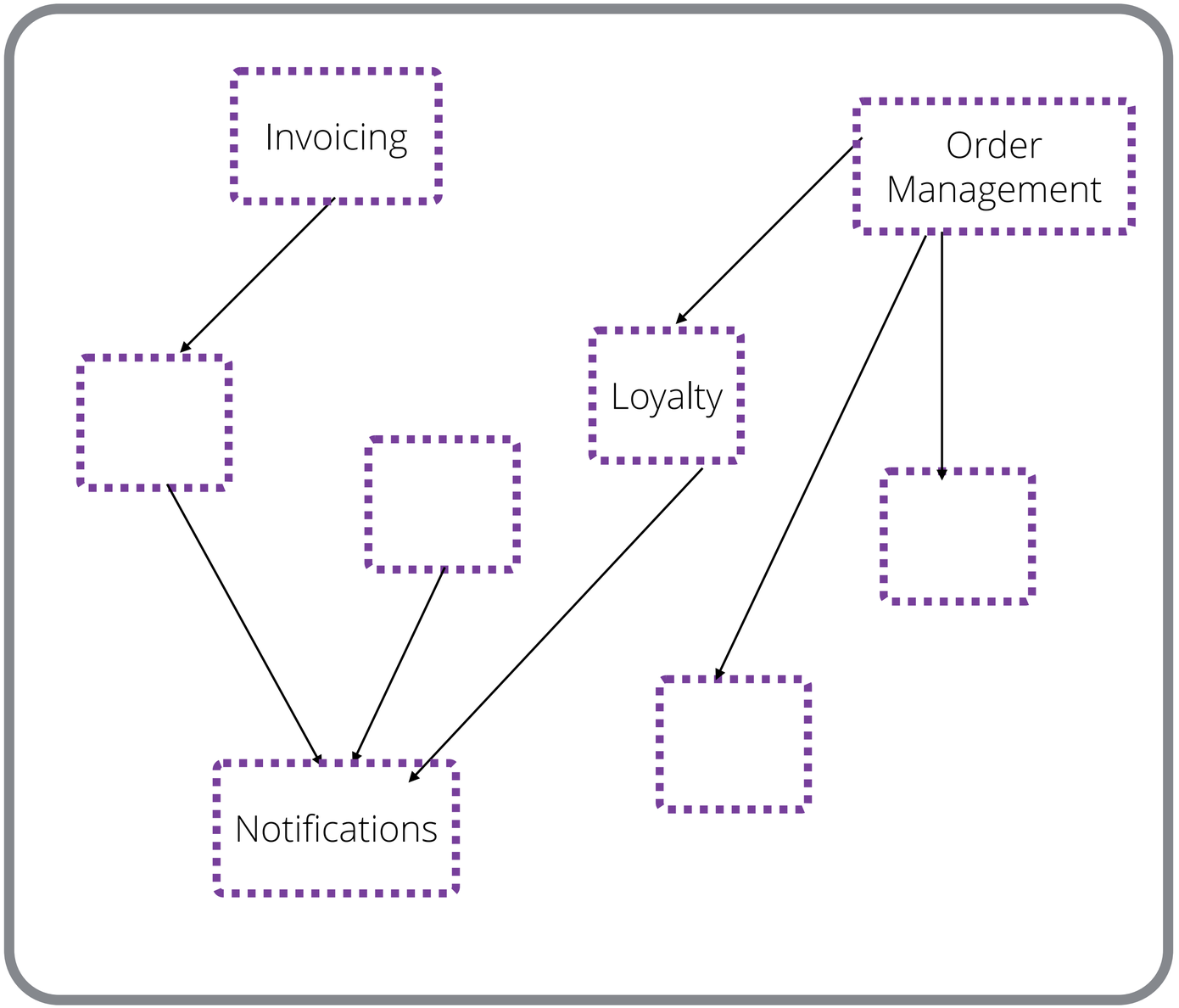
## GETTING FINER-GRAINED

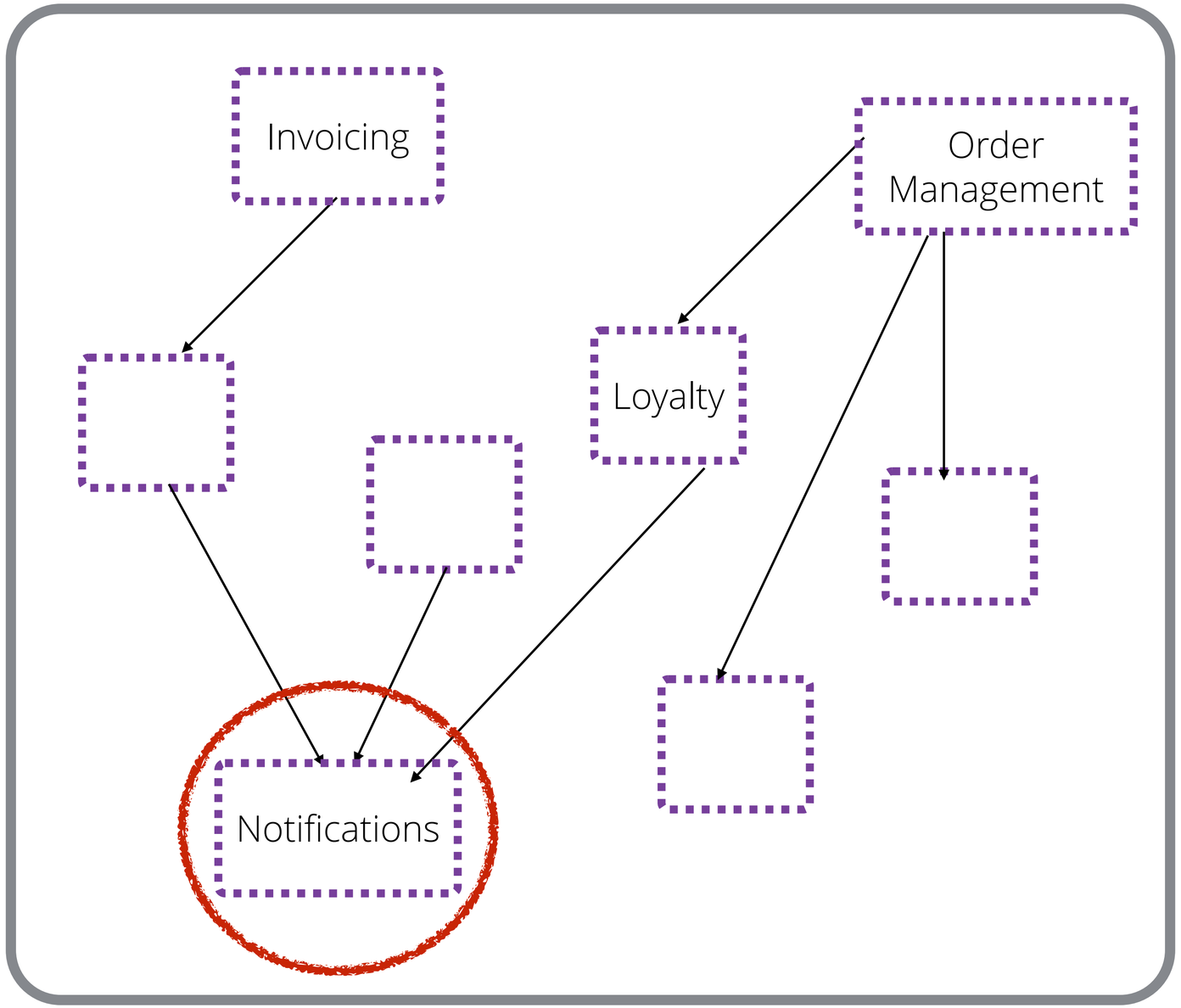
**Can break down around aggregate boundaries**

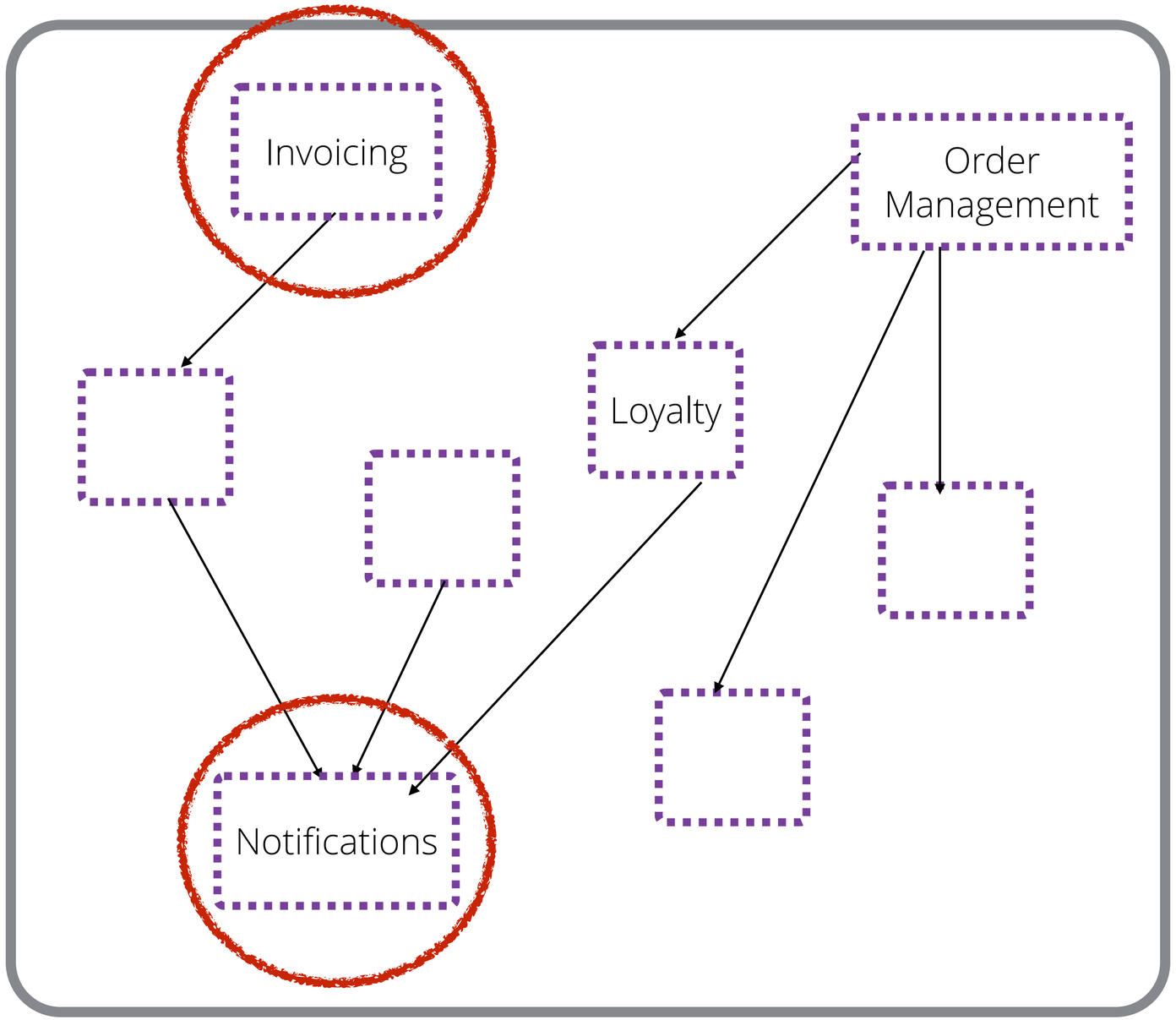
**Keep the higher-level abstraction for outside parties**



# 3/4 Planning a migration


Invoicing
Order Management
Loyalty
Notifications


Invoicing
Order Management
Loyalty
Notifications


Invoicing
Order Management
Loyalty
Notifications

# Remember your goal?

# Scaling system

## COMPETING FORCES

Scaling system

Speeding up delivery

## COMPETING FORCES

Scaling system

Speeding up delivery

Changing technology

## COMPETING FORCES

Scaling system

Speeding up delivery

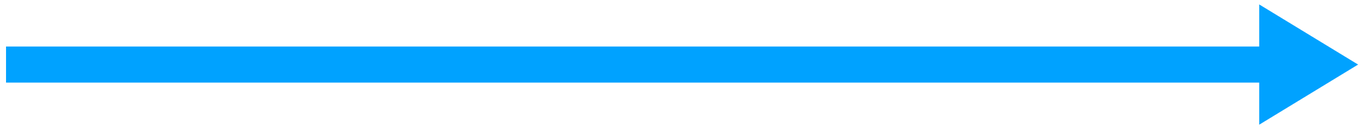
Changing technology

Improving robustness

# IT’S ALL ABOUT TRADEOFFS

Achieving your goal

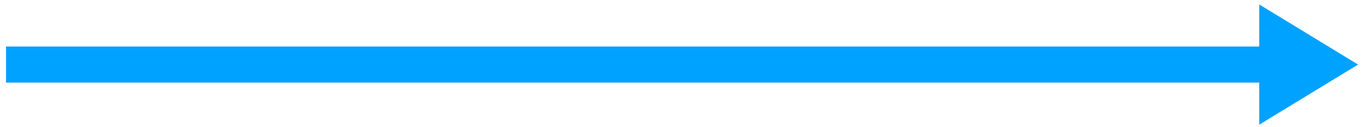
# IT’S ALL ABOUT TRADEOFFS



Achieving your goal

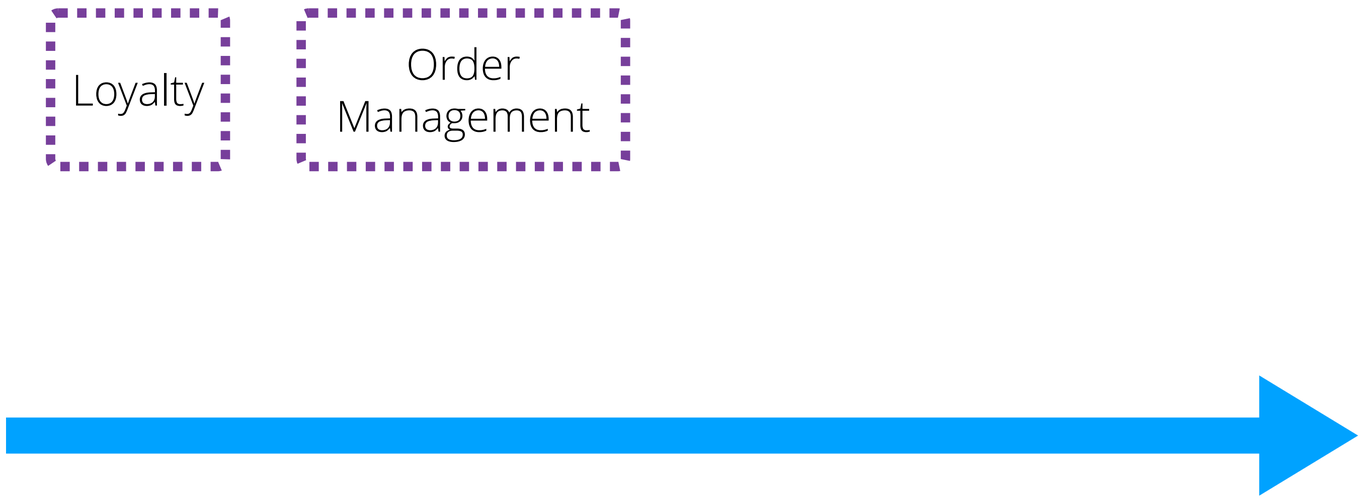
# IT’S ALL ABOUT TRADEOFFS

Loyalty

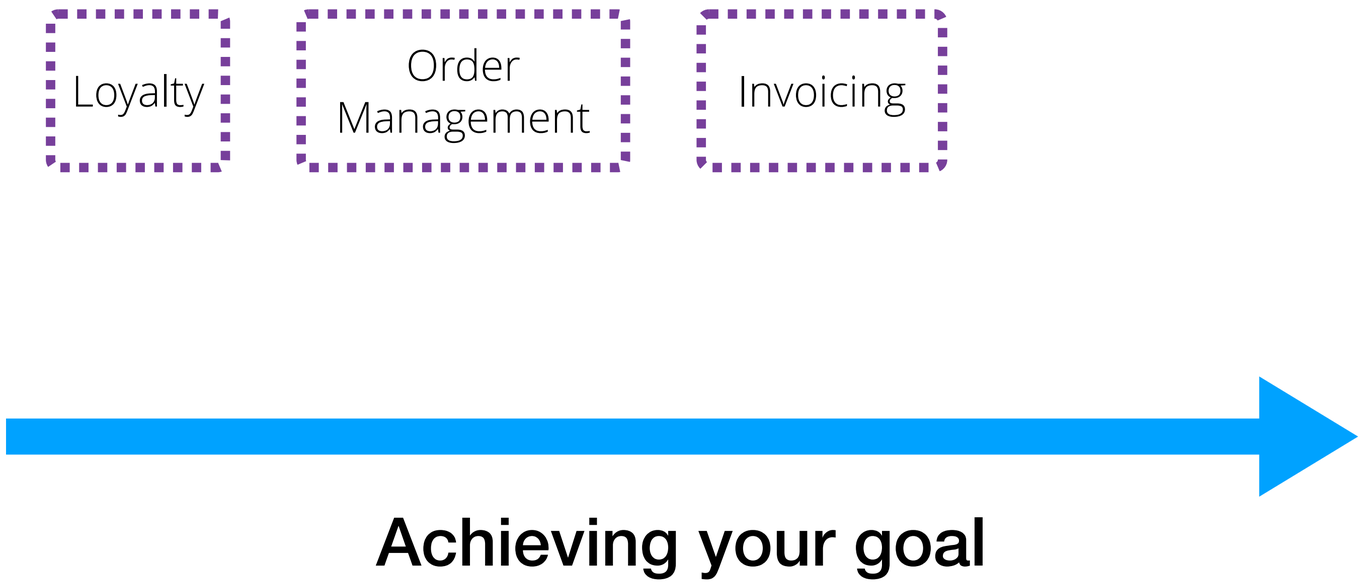


Achieving your goal

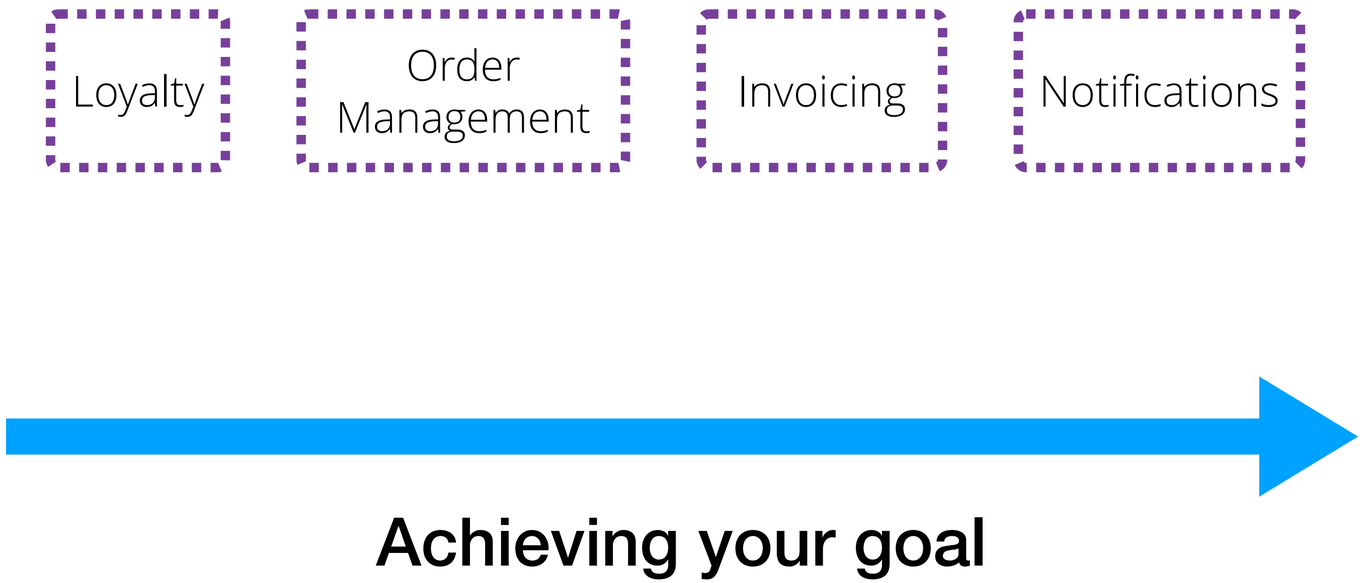
# IT'S ALL ABOUT TRADEOFFS



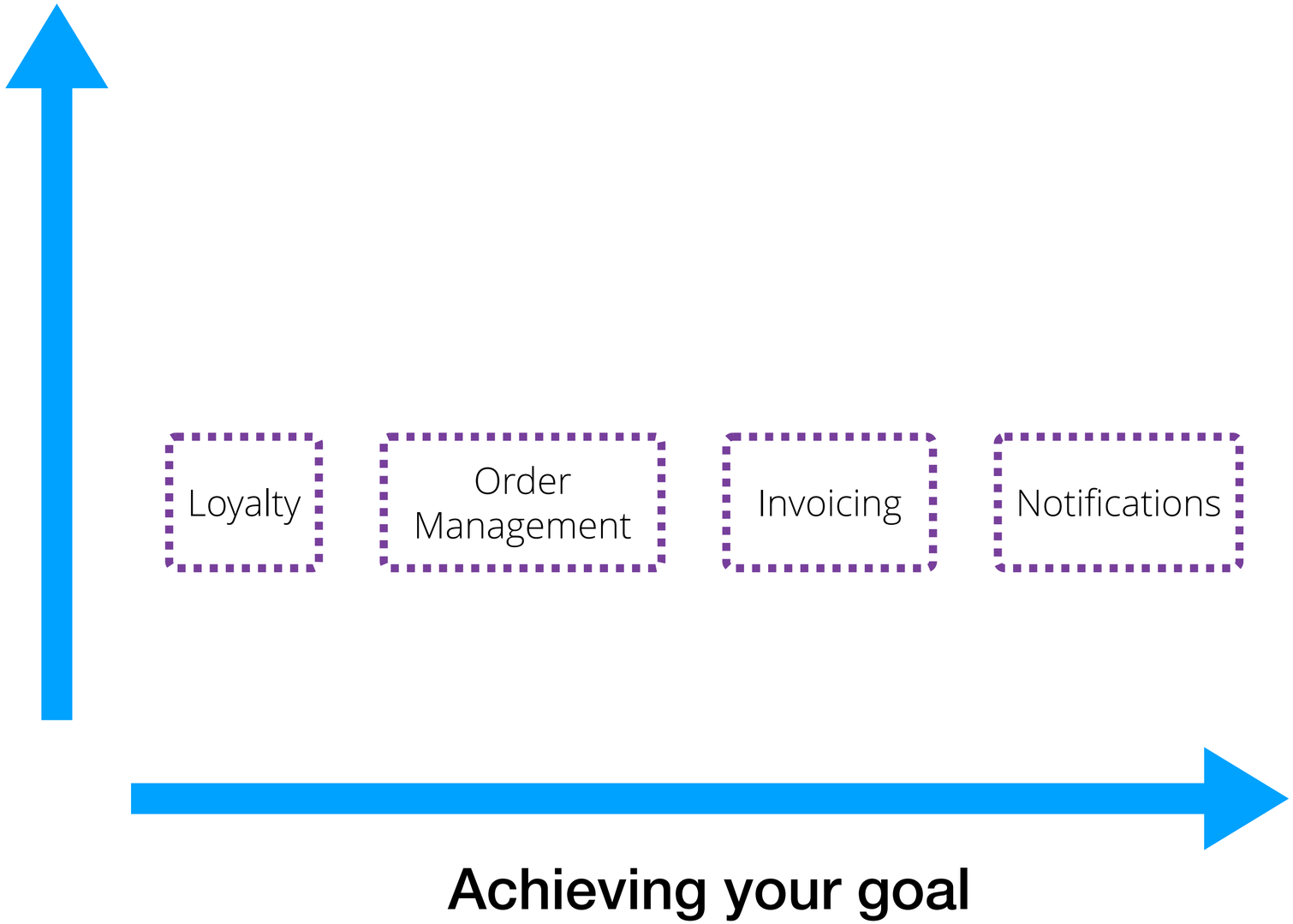
# IT'S ALL ABOUT TRADEOFFS



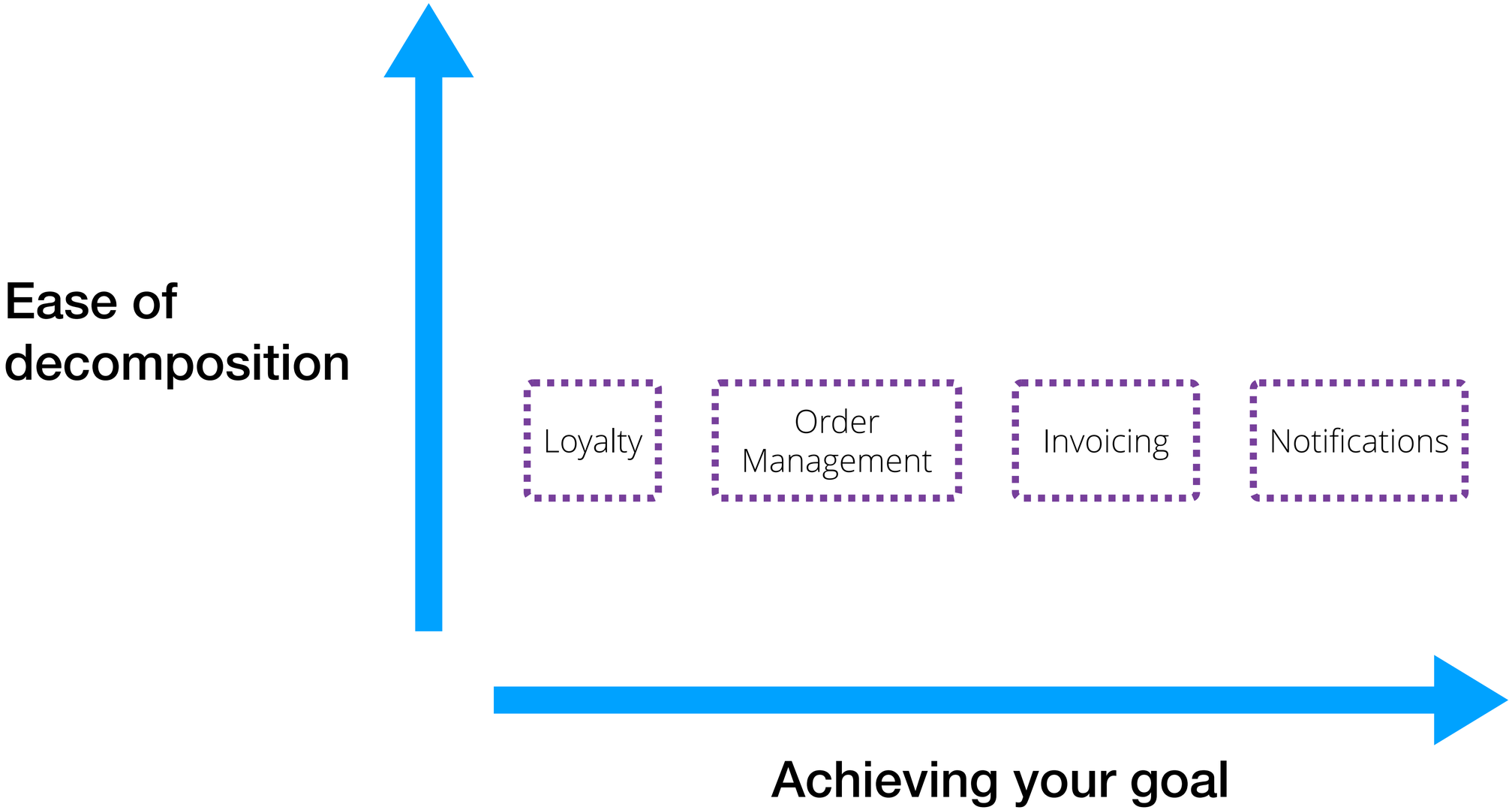
# IT'S ALL ABOUT TRADEOFFS



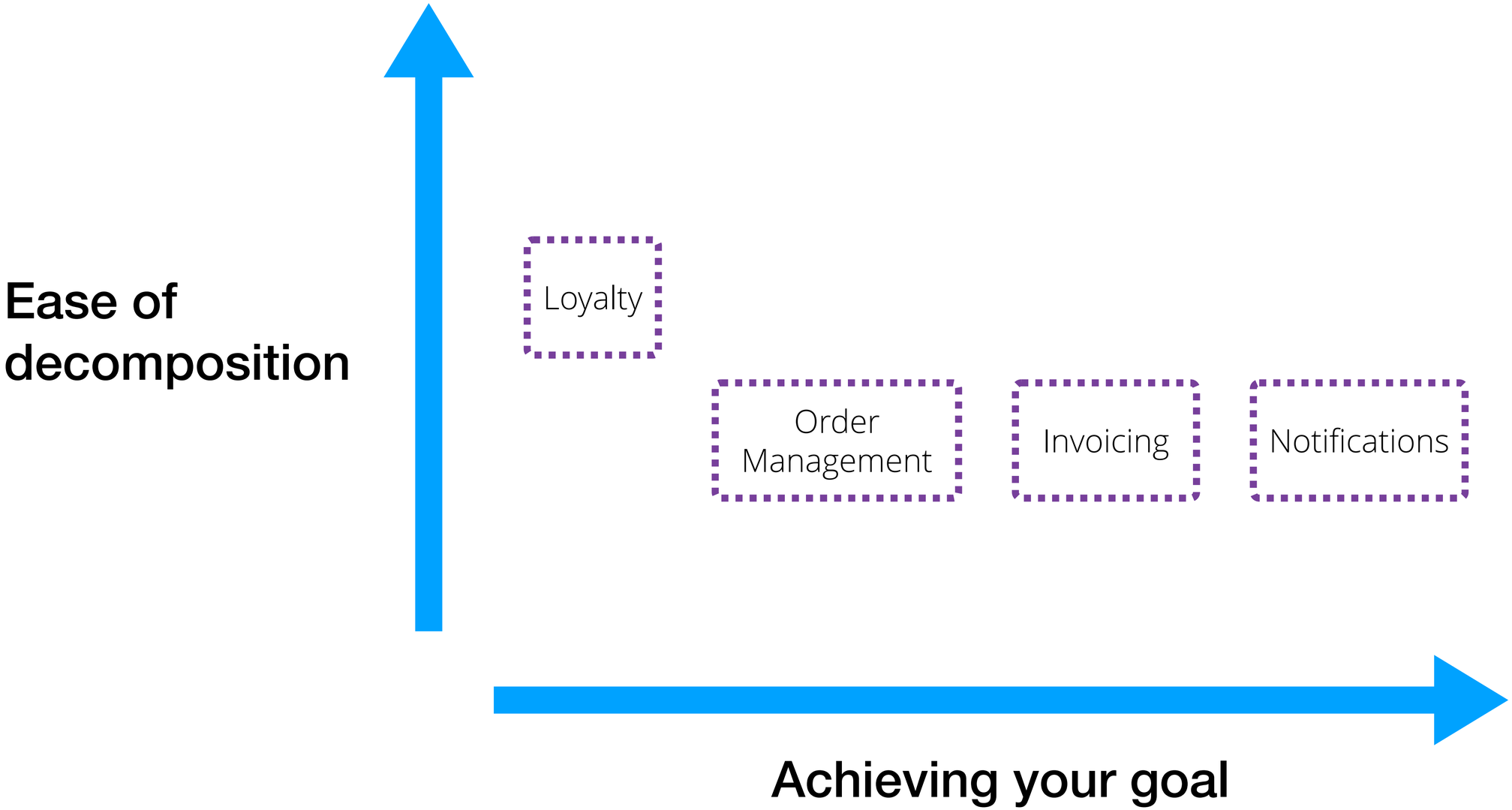
# IT'S ALL ABOUT TRADEOFFS



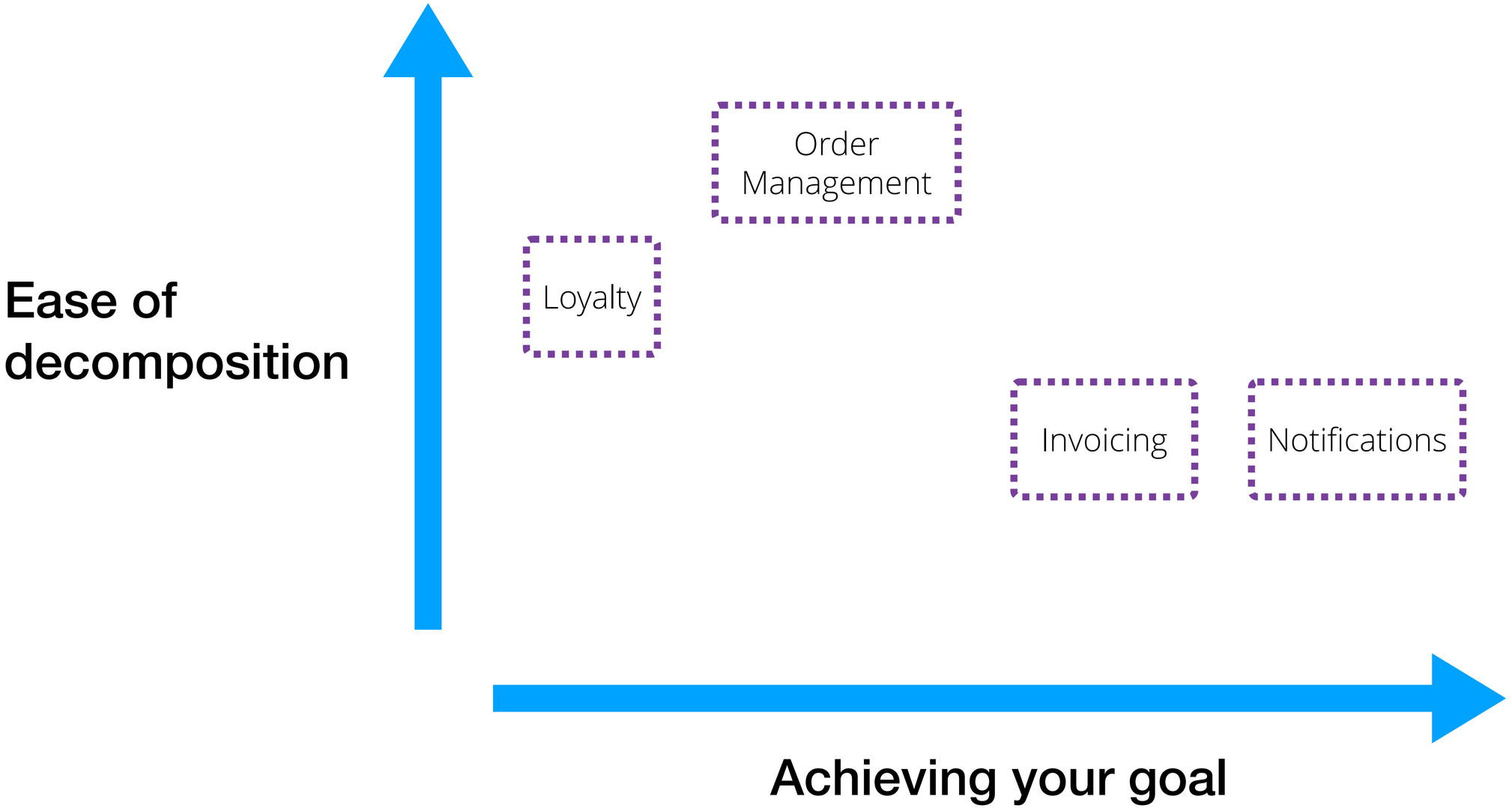
# IT'S ALL ABOUT TRADEOFFS



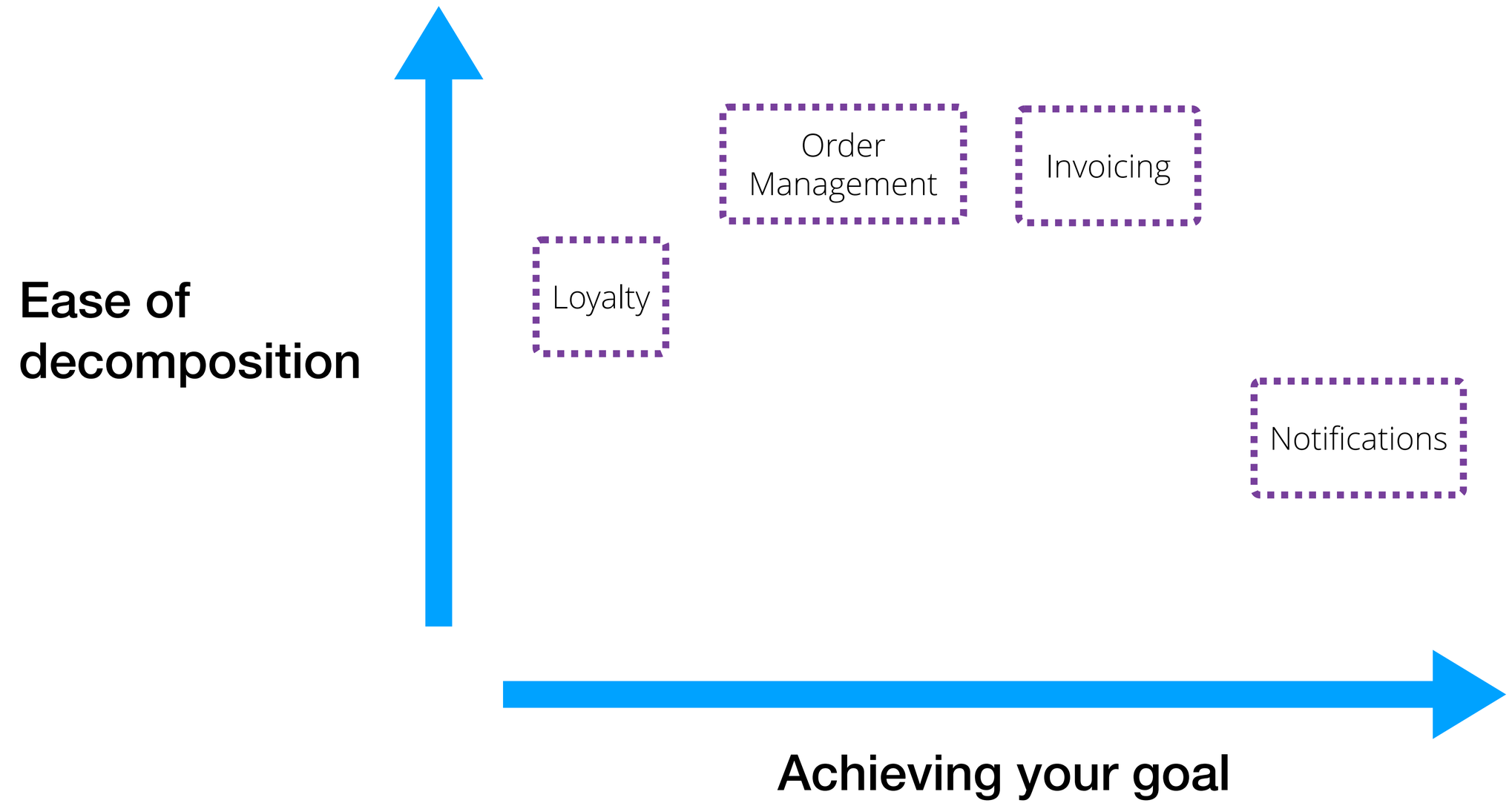
# IT'S ALL ABOUT TRADEOFFS



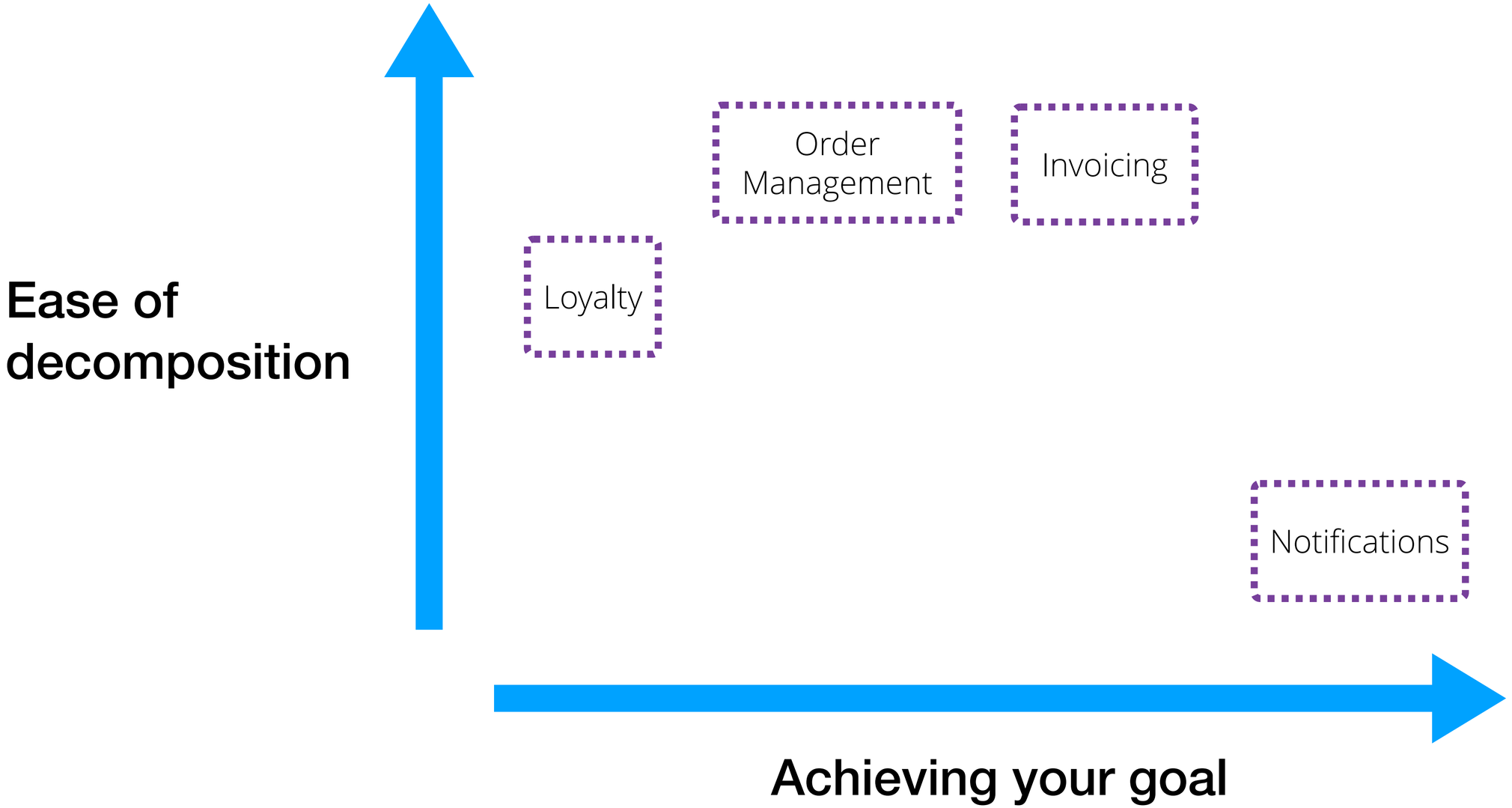
# IT'S ALL ABOUT TRADEOFFS



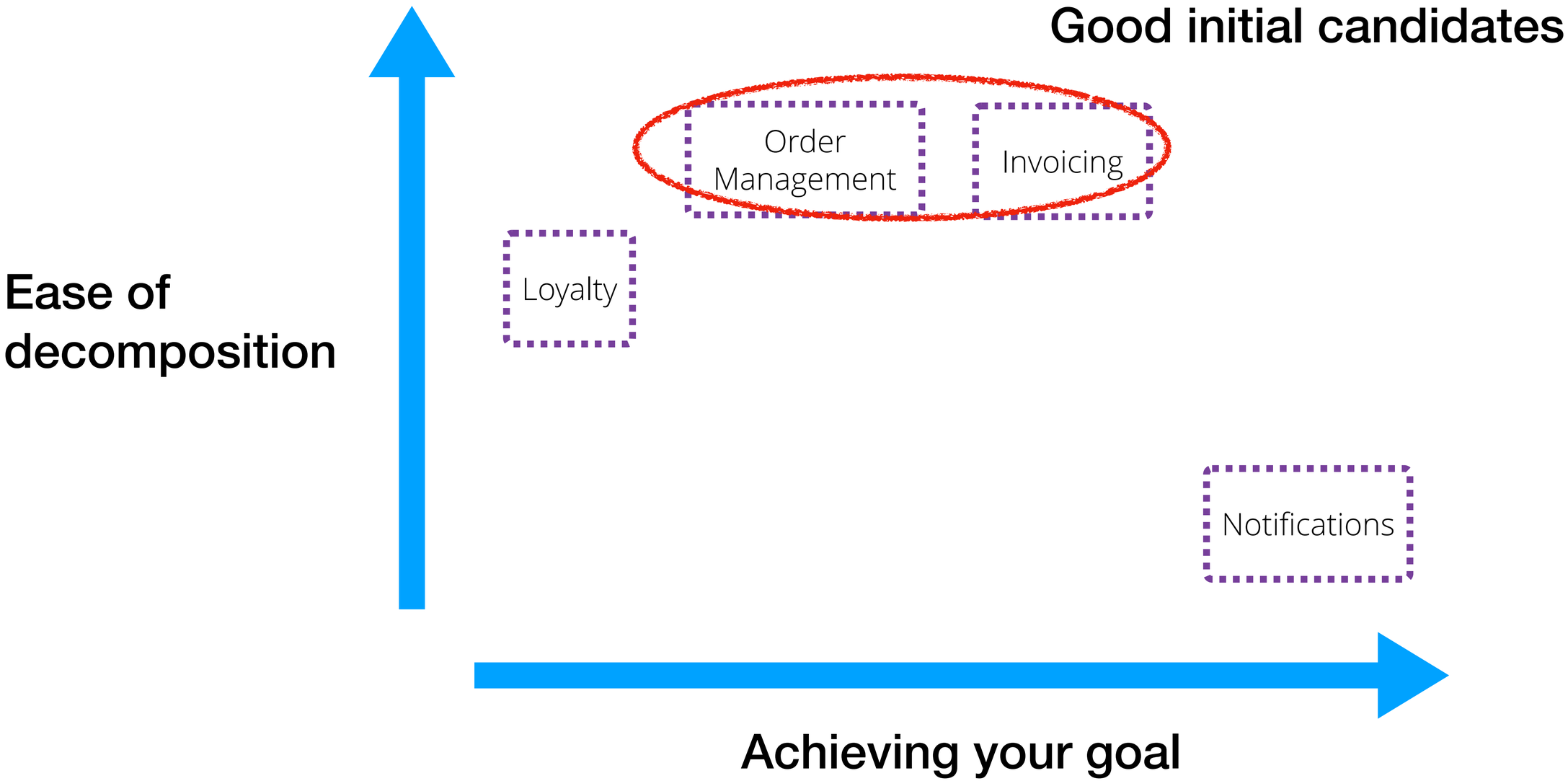
# IT'S ALL ABOUT TRADEOFFS



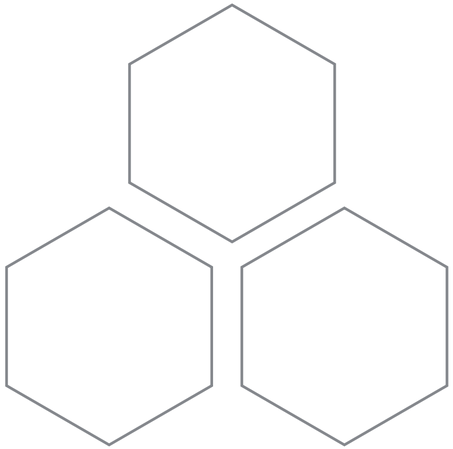
# IT'S ALL ABOUT TRADEOFFS

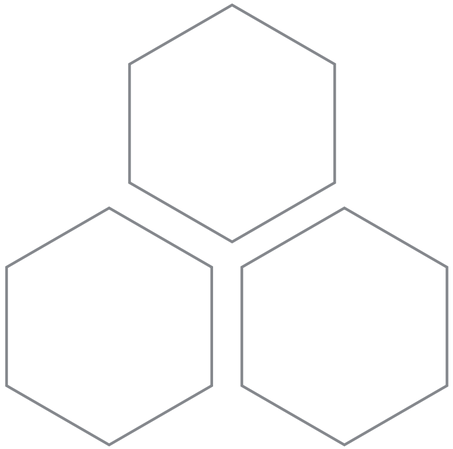


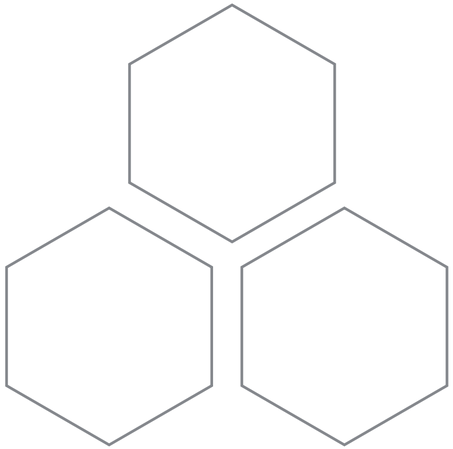
# IT'S ALL ABOUT TRADEOFFS

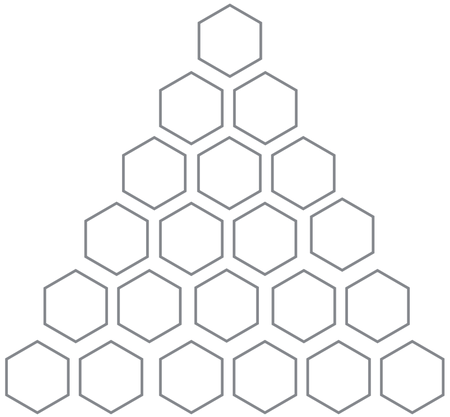



https://www.flickr.com/photos/orinrobertjohn/2911248047/

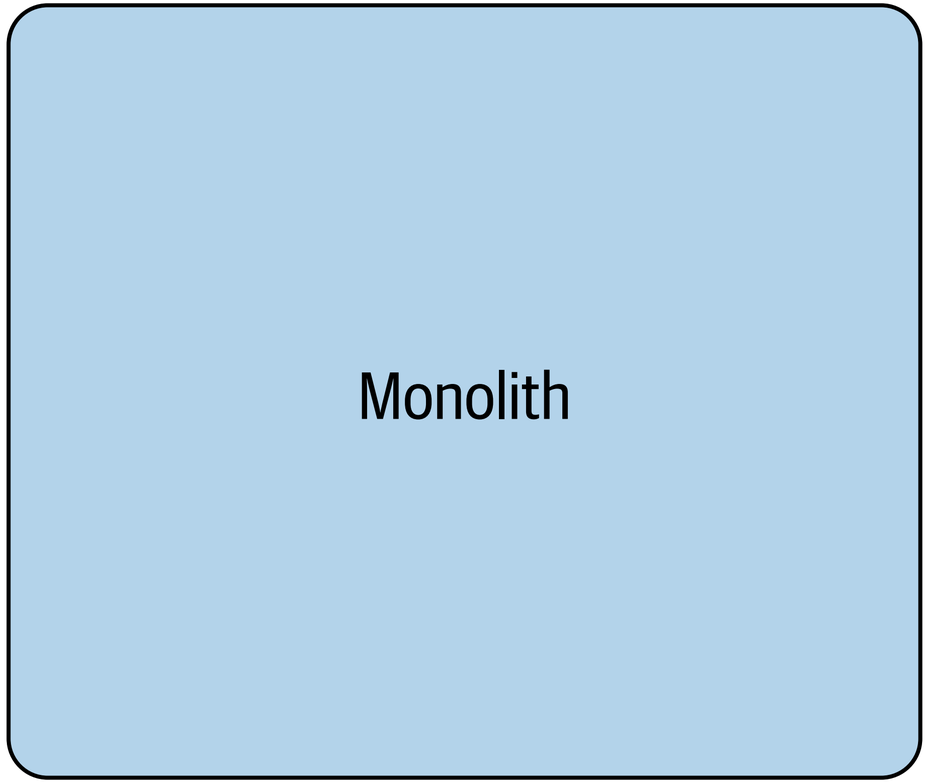




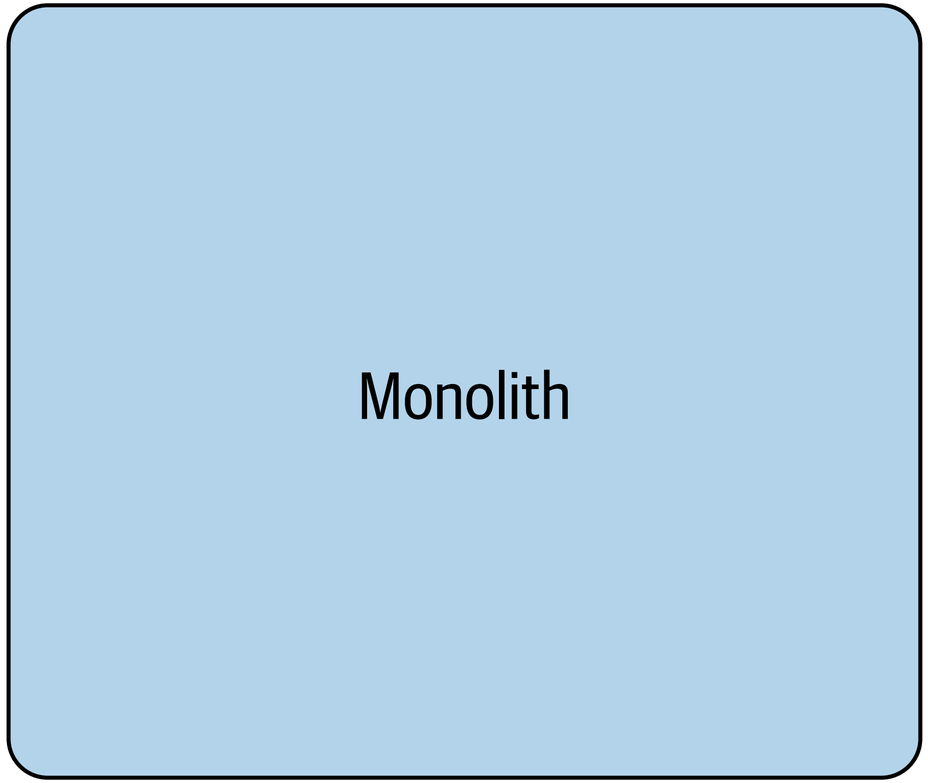


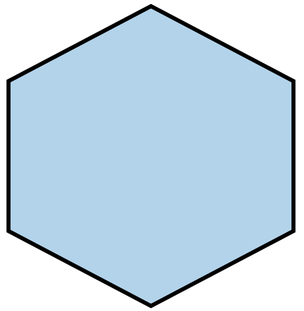


# INCREMENTAL DECOMPOSITION FTW

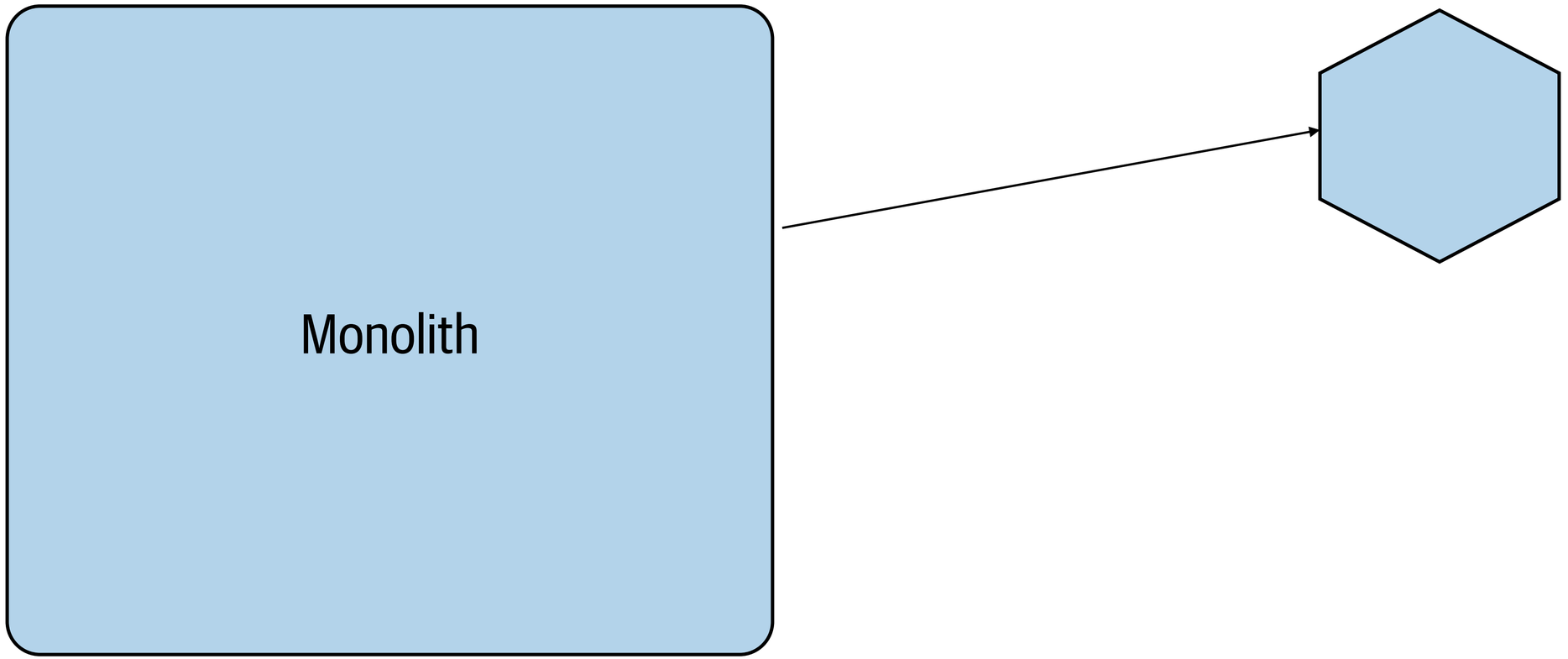


# INCREMENTAL DECOMPOSITION FTW

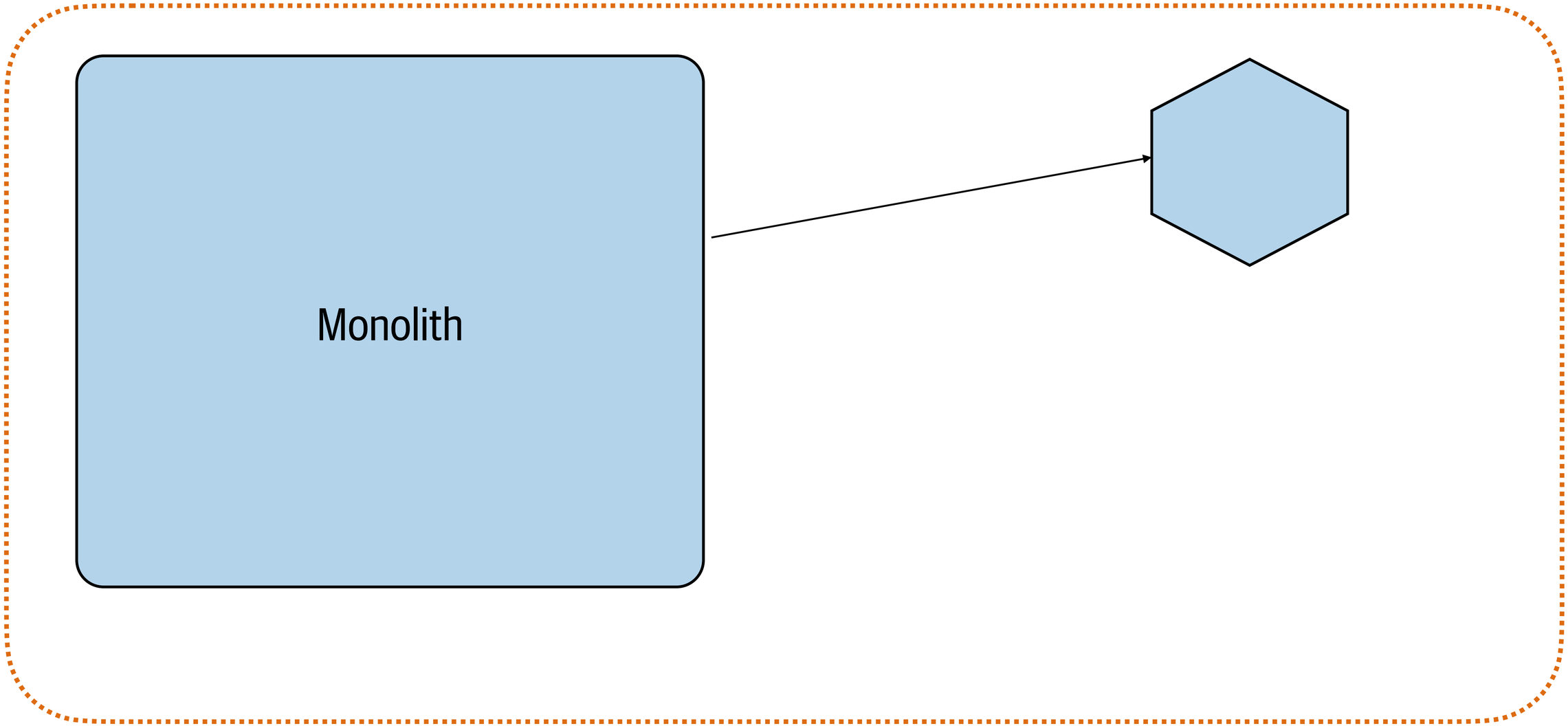


# INCREMENTAL DECOMPOSITION FTW



# INCREMENTAL DECOMPOSITION FTW



You won’t appreciate the true horror, pain and suffering of microservices until you’re running them in production

“If you do a big bang rewrite, the only thing you’re certain of is a big bang”

- Martin Fowler (paraphrased)

# Move functionality to microservices a piece at a time

Move functionality to microservices a piece at a time

Get it into production to start getting value from it, and learning from the experience

# 4/4 Application Decomposition Patterns

# PATTERN: STRANGLER FIG

By Poyt448 Peter Woodard - Own work, Public Domain, https://commons.wikimedia.org/w/index.php?curid=5920414

# PATTERN: STRANGLER FIG

By Poyt448 Peter Woodard - Own work, Public Domain, https://commons.wikimedia.org/w/index.php?curid=5920414

Wrap a new system around the old system

# PATTERN: STRANGLER FIG

By Poyt448 Peter Woodard - Own work, Public Domain, https://commons.wikimedia.org/w/index.php?curid=5920414

Wrap a new system around the old system

The new system doesn’t need to fully replace the old

# PATTERN: STRANGLER FIG

By Poyt448 Peter Woodard - Own work, Public Domain, https://commons.wikimedia.org/w/index.php?curid=5920414

Wrap a new system around the old system

The new system doesn't need to fully replace the old

Calls to functionality in the new system are intercepted, other calls are served by the old system

## STRANGLER FIG IMPLEMENTATION

Intercept calls on permitter of old system (the monolith)

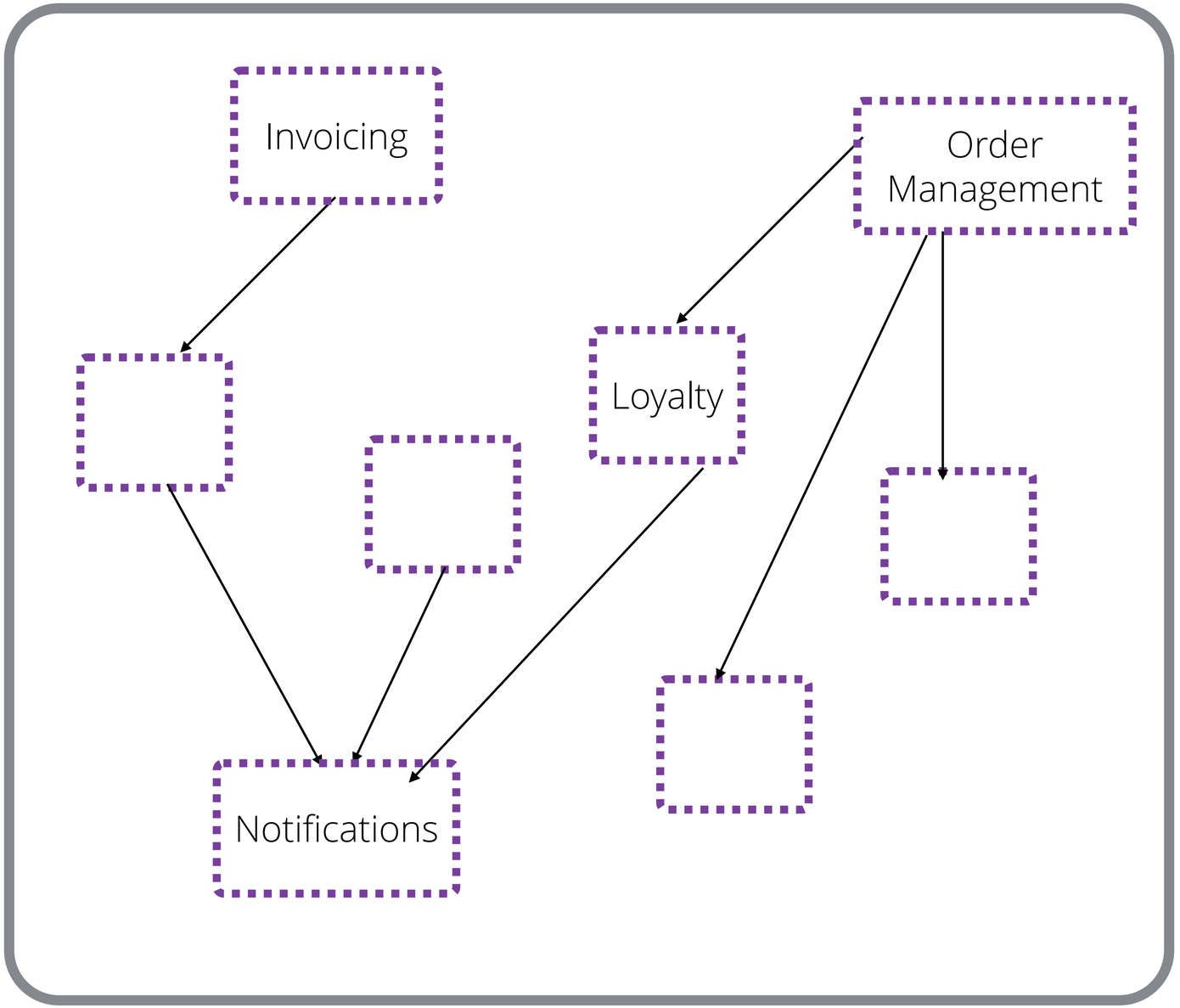
## STRANGLER FIG IMPLEMENTATION

Intercept calls on permitter of old system (the monolith)

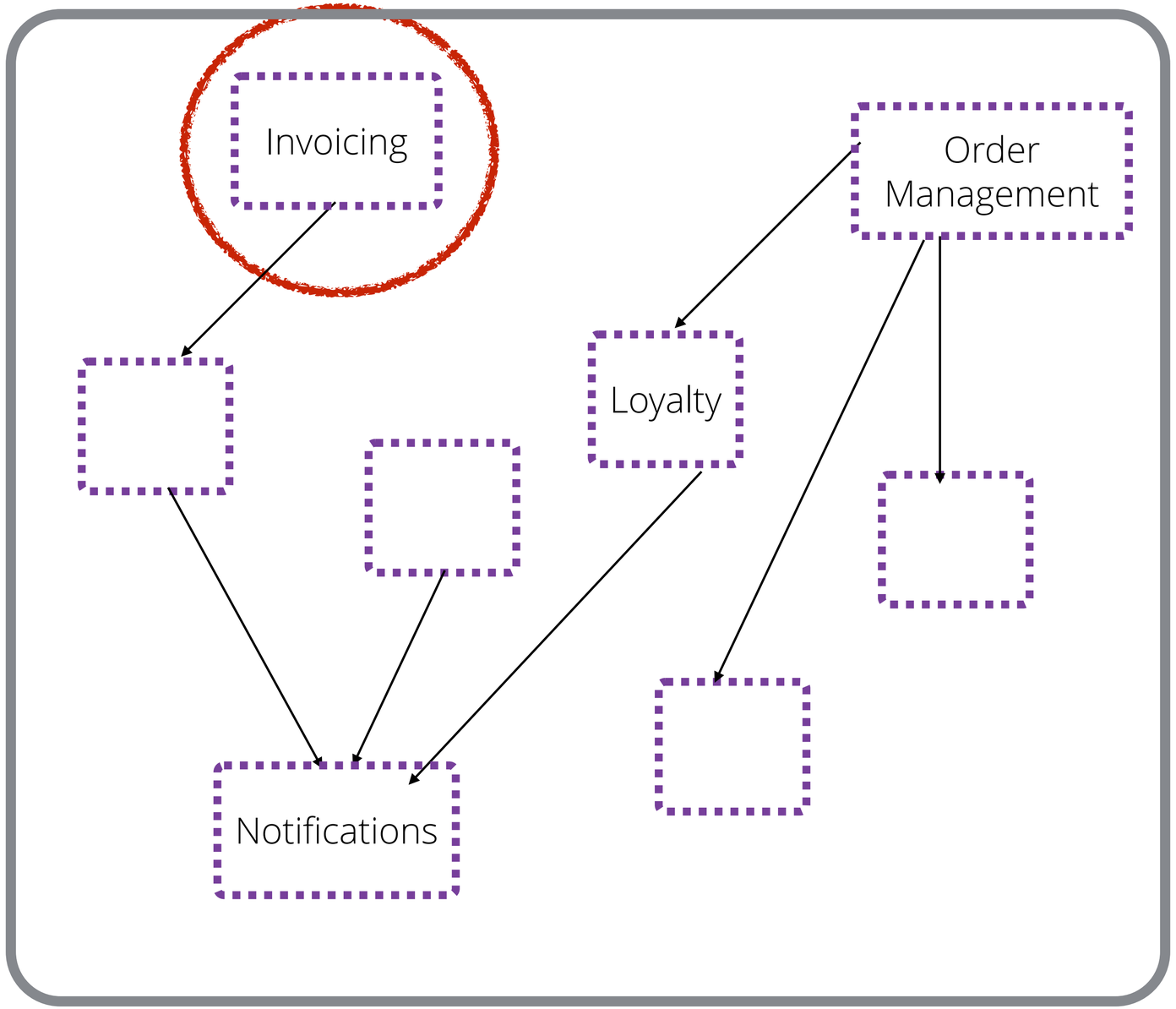
Divert calls to where the functionality is located

# CANDIDATES

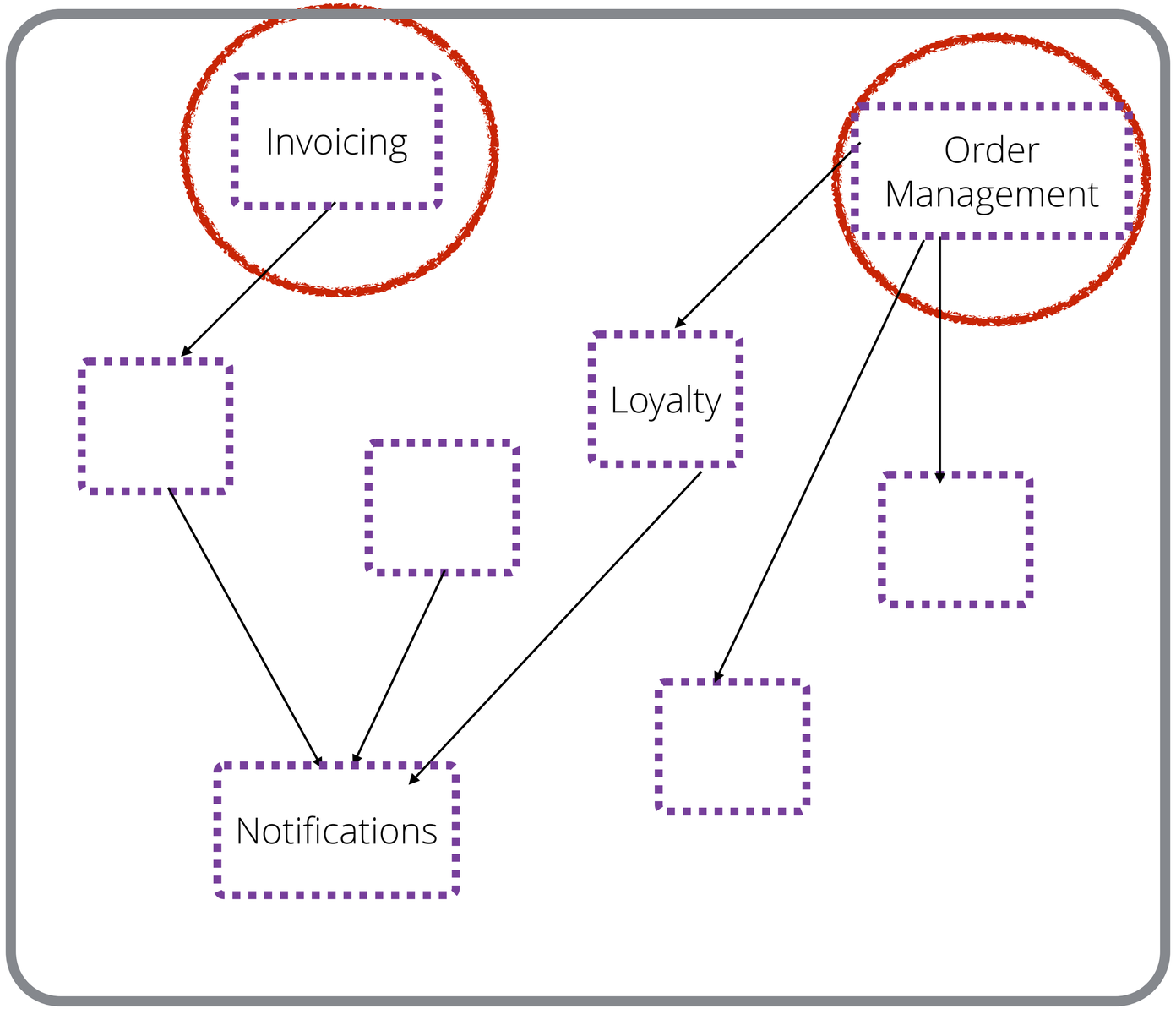
# CANDIDATES



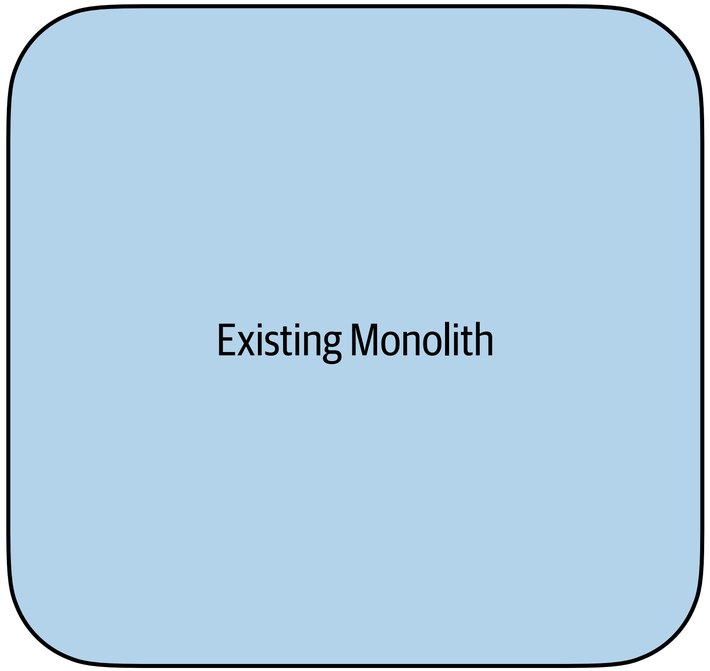
# CANDIDATES



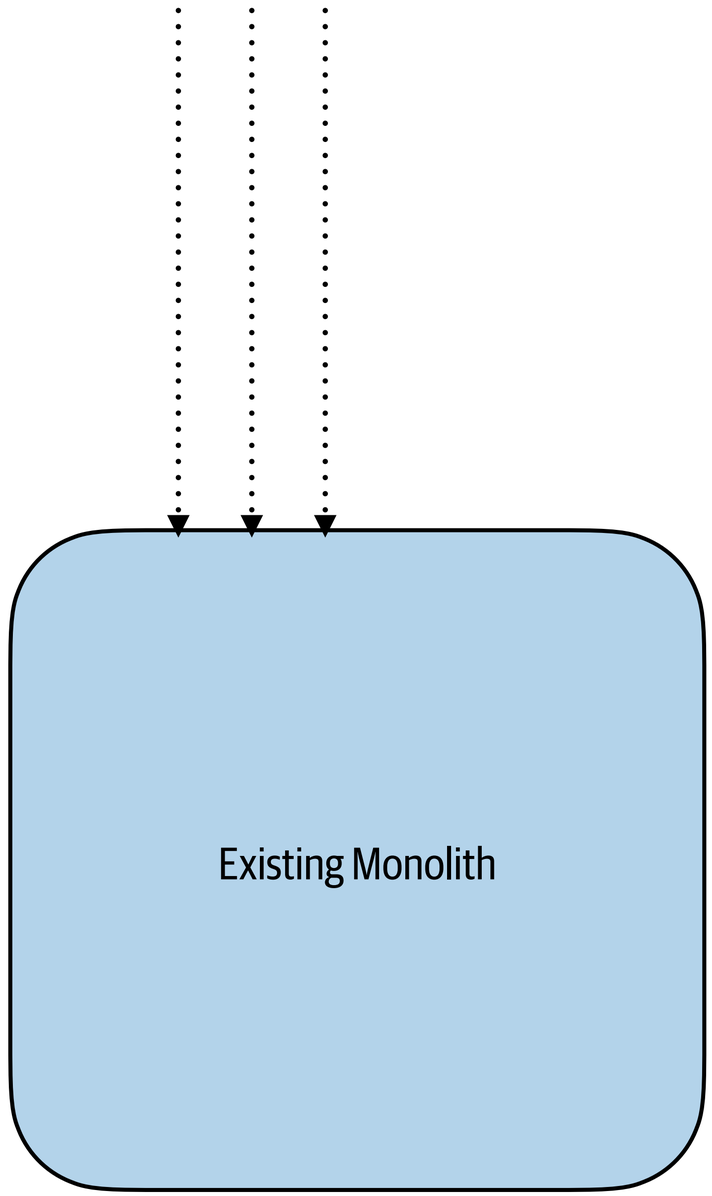
# CANDIDATES



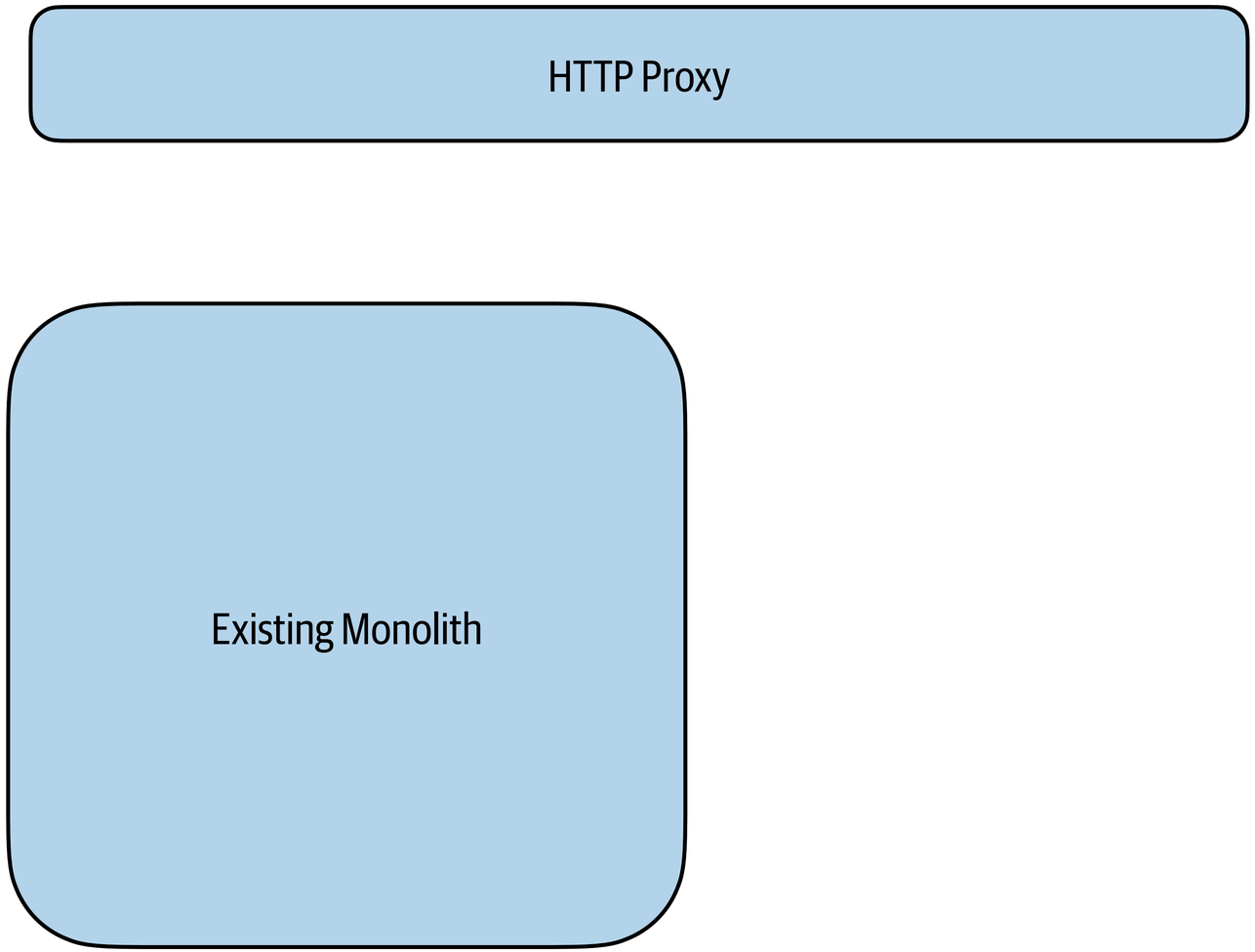
# INTERCEPTION



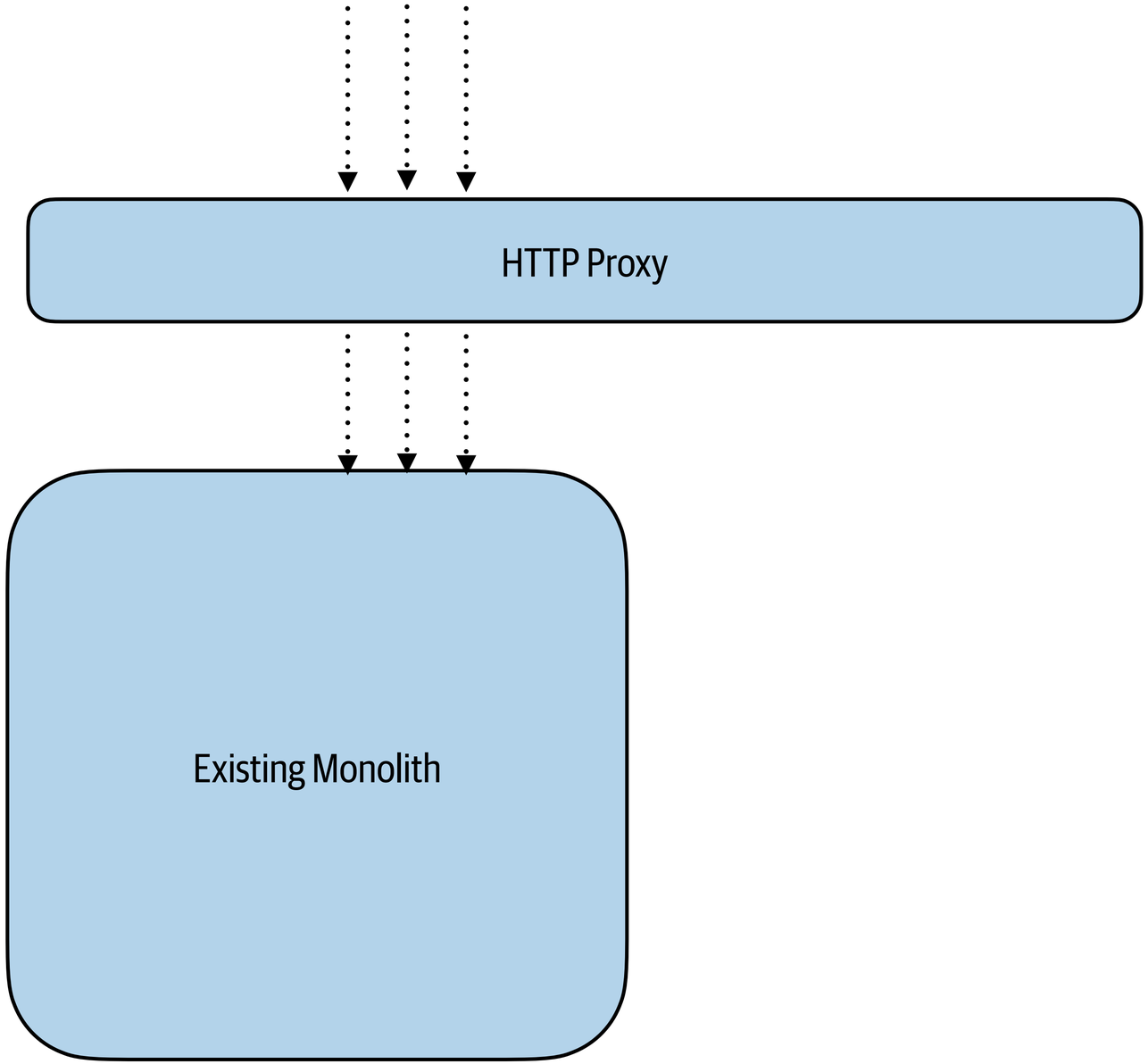
# INTERCEPTION



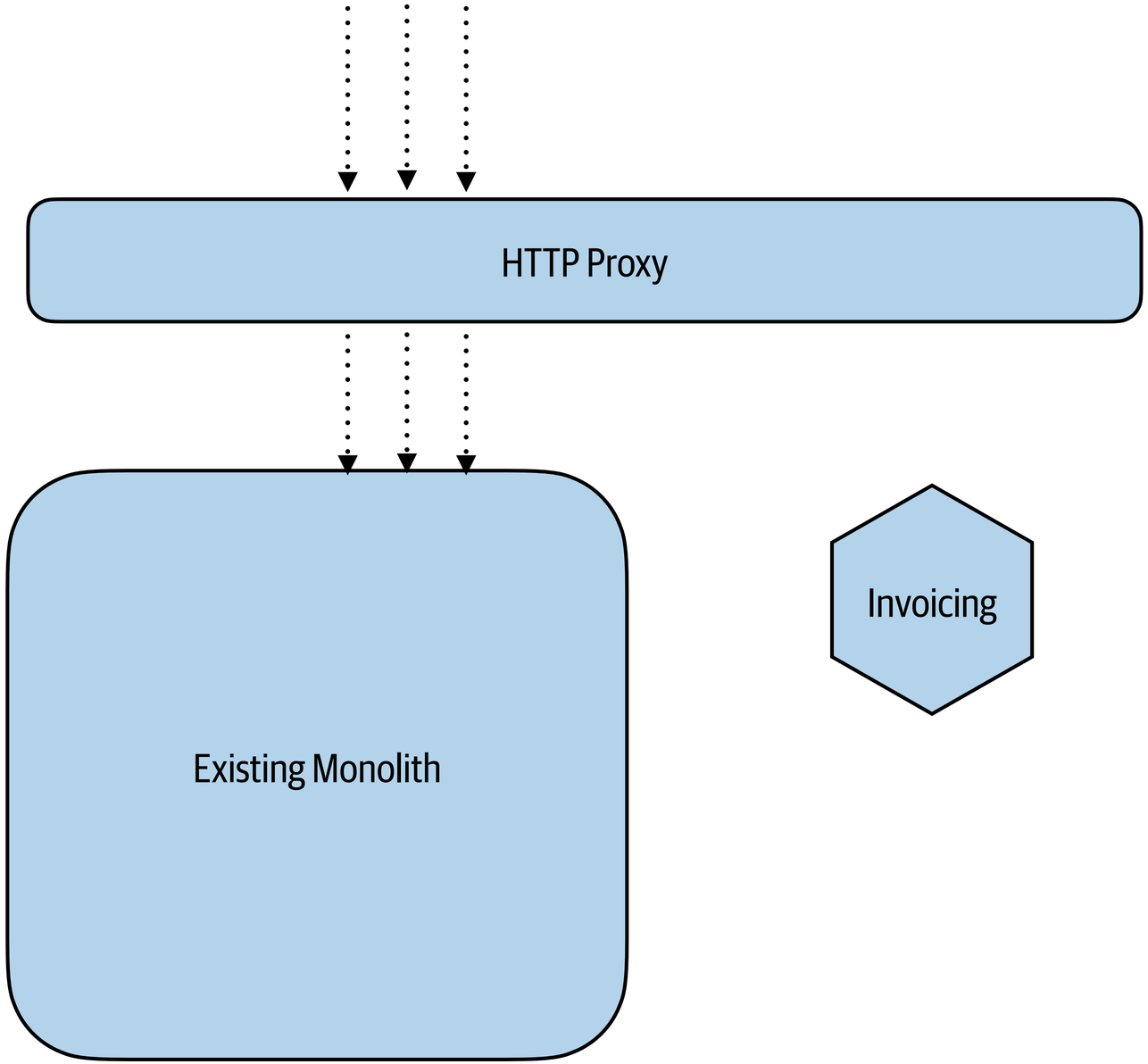
# INTERCEPTION



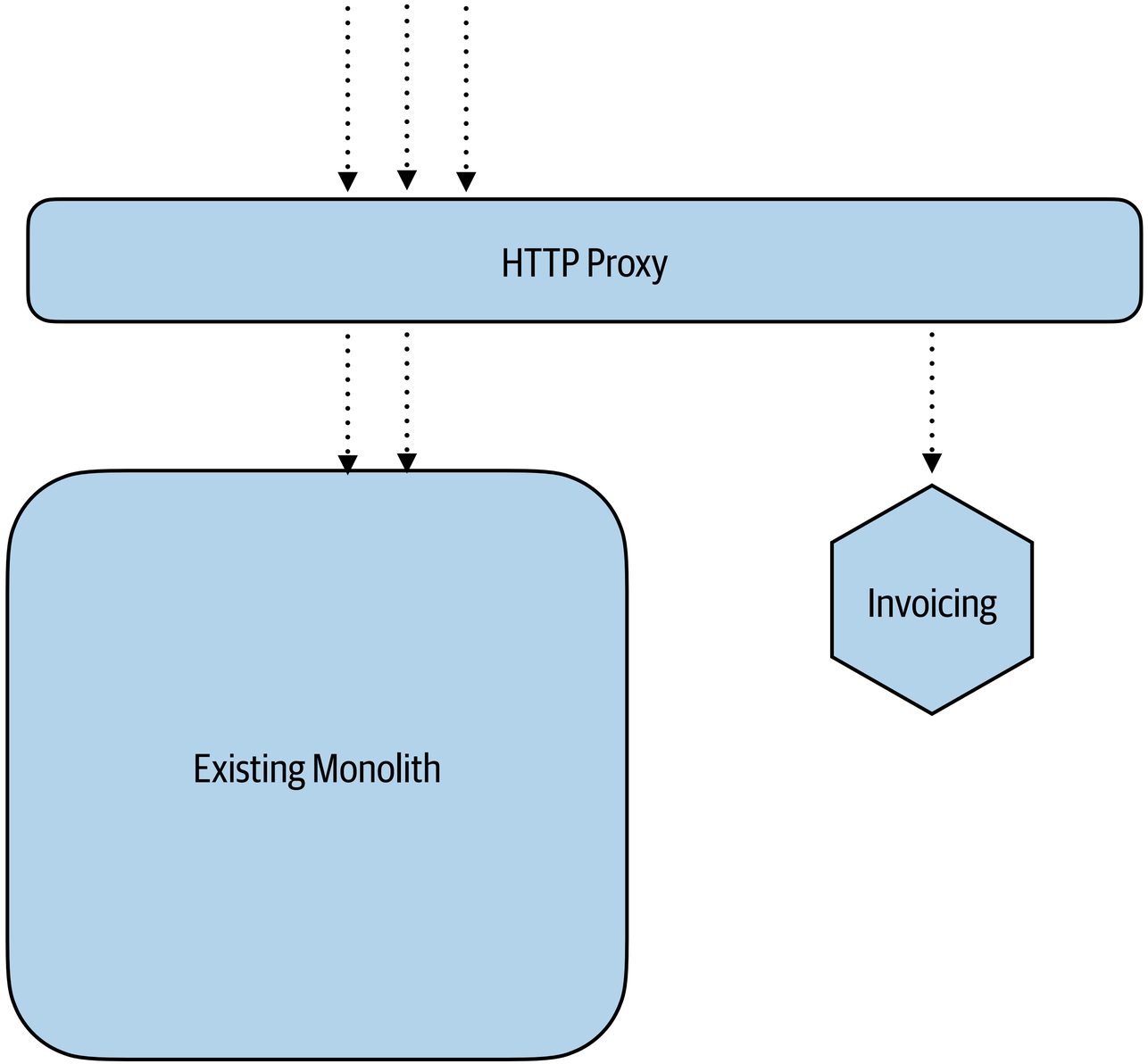
# INTERCEPTION



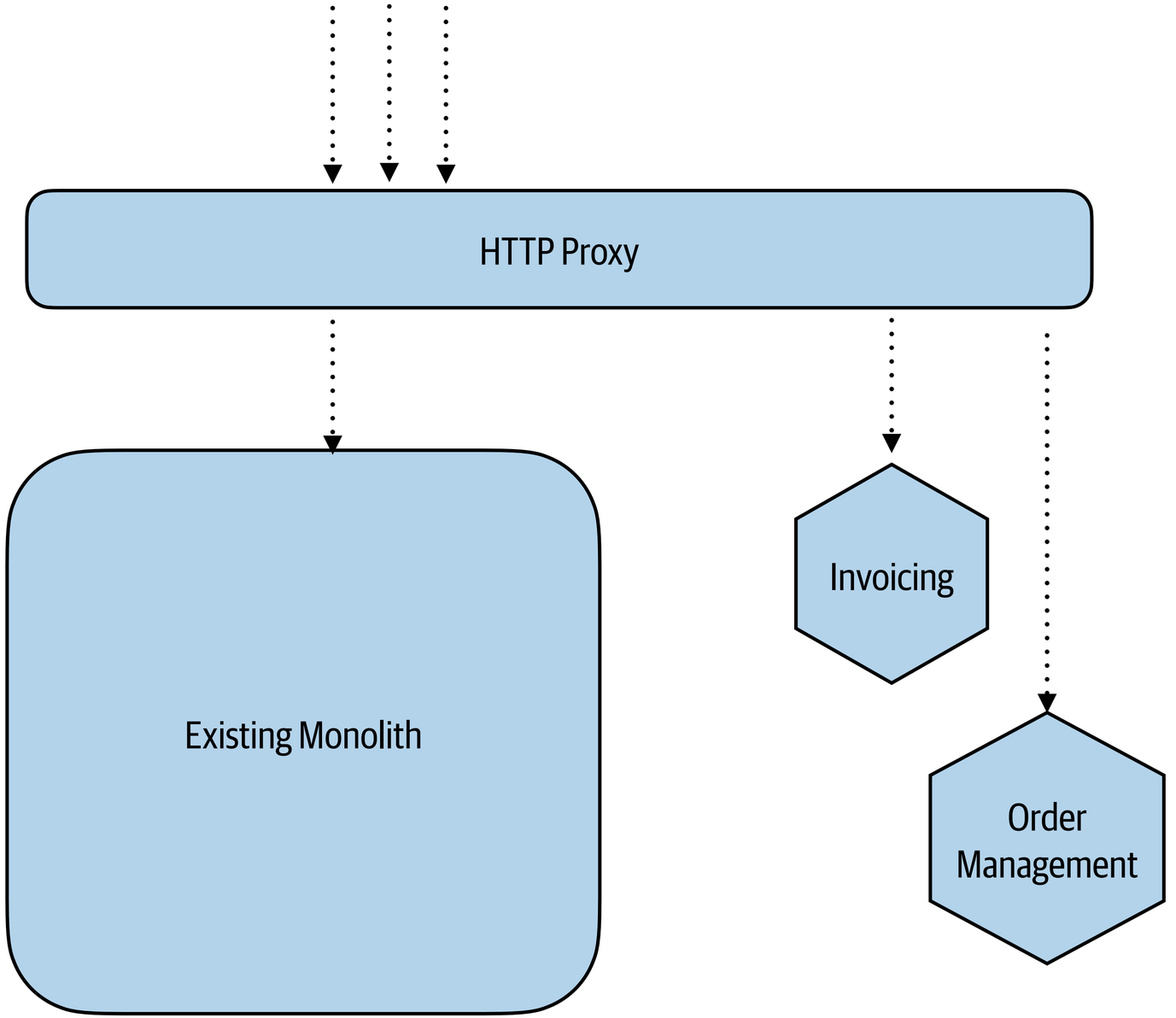
# INTERCEPTION



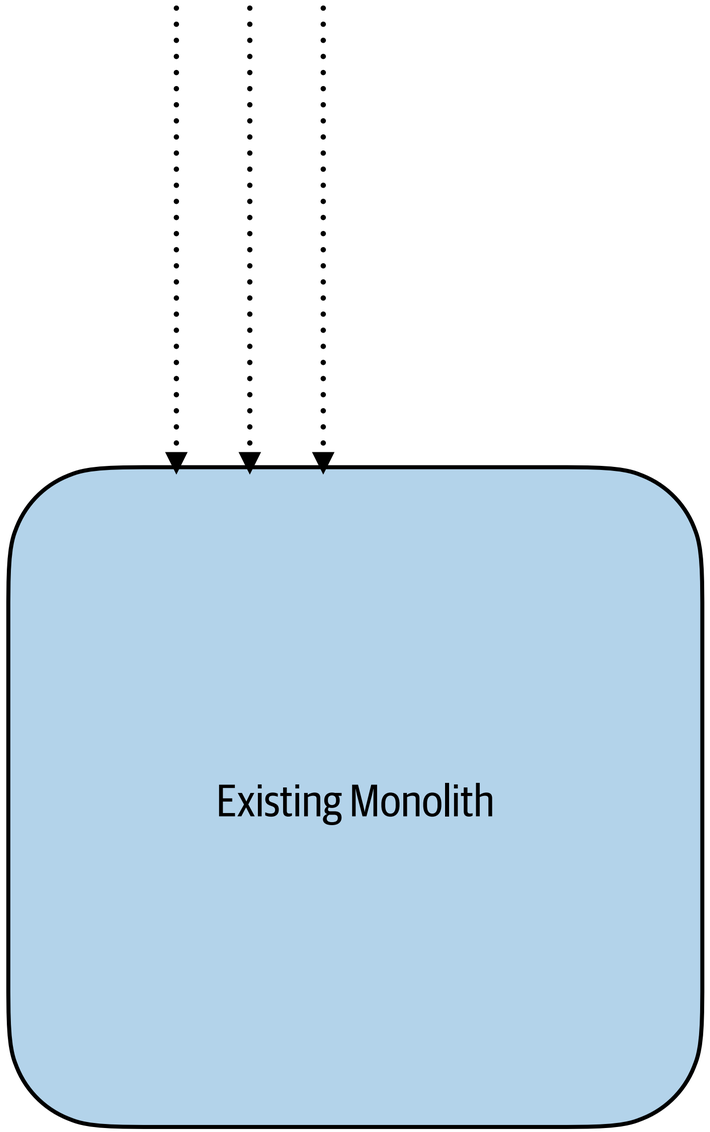
# INTERCEPTION



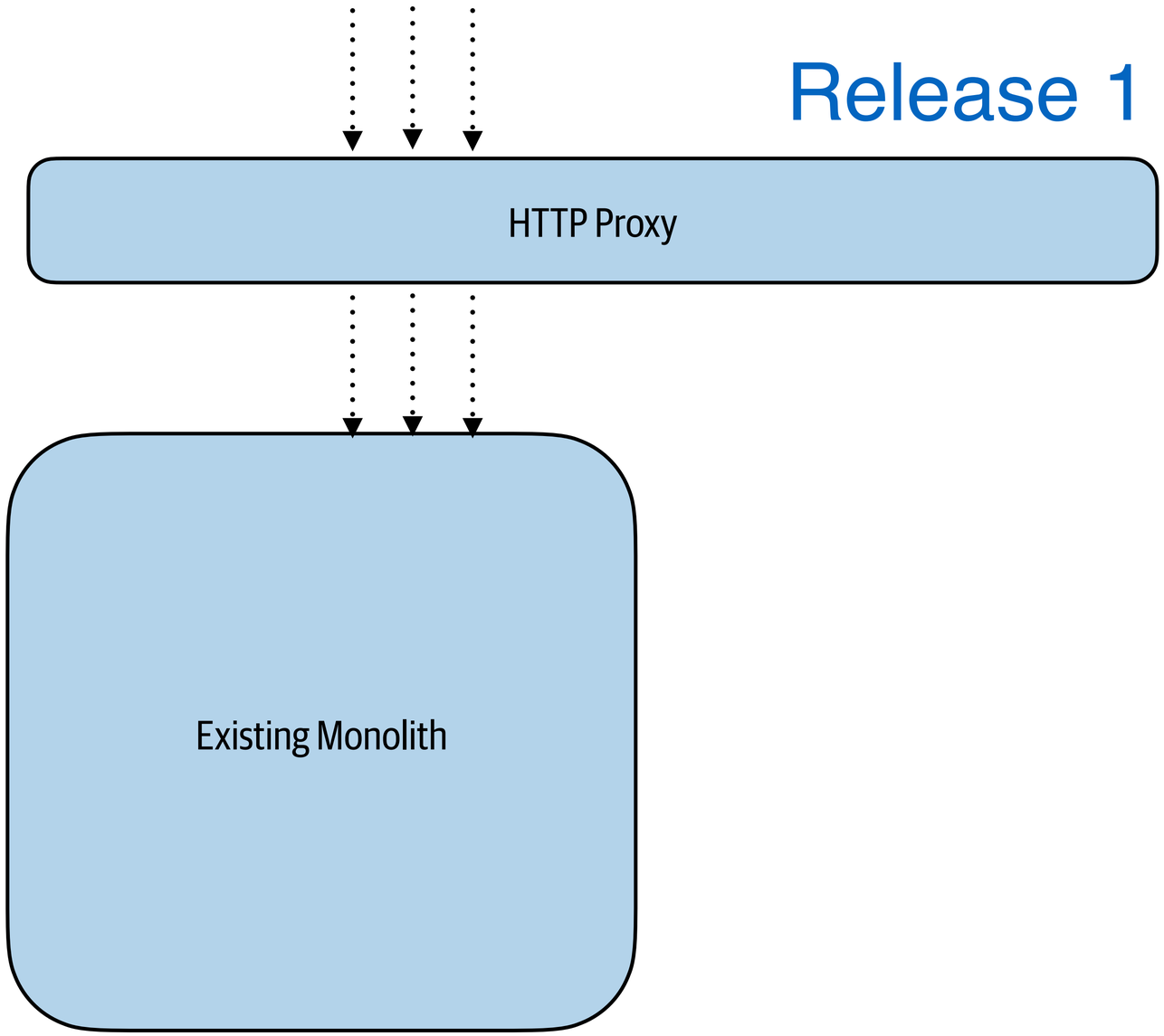
# INTERCEPTION



# INCREMENTAL ROLLOUT

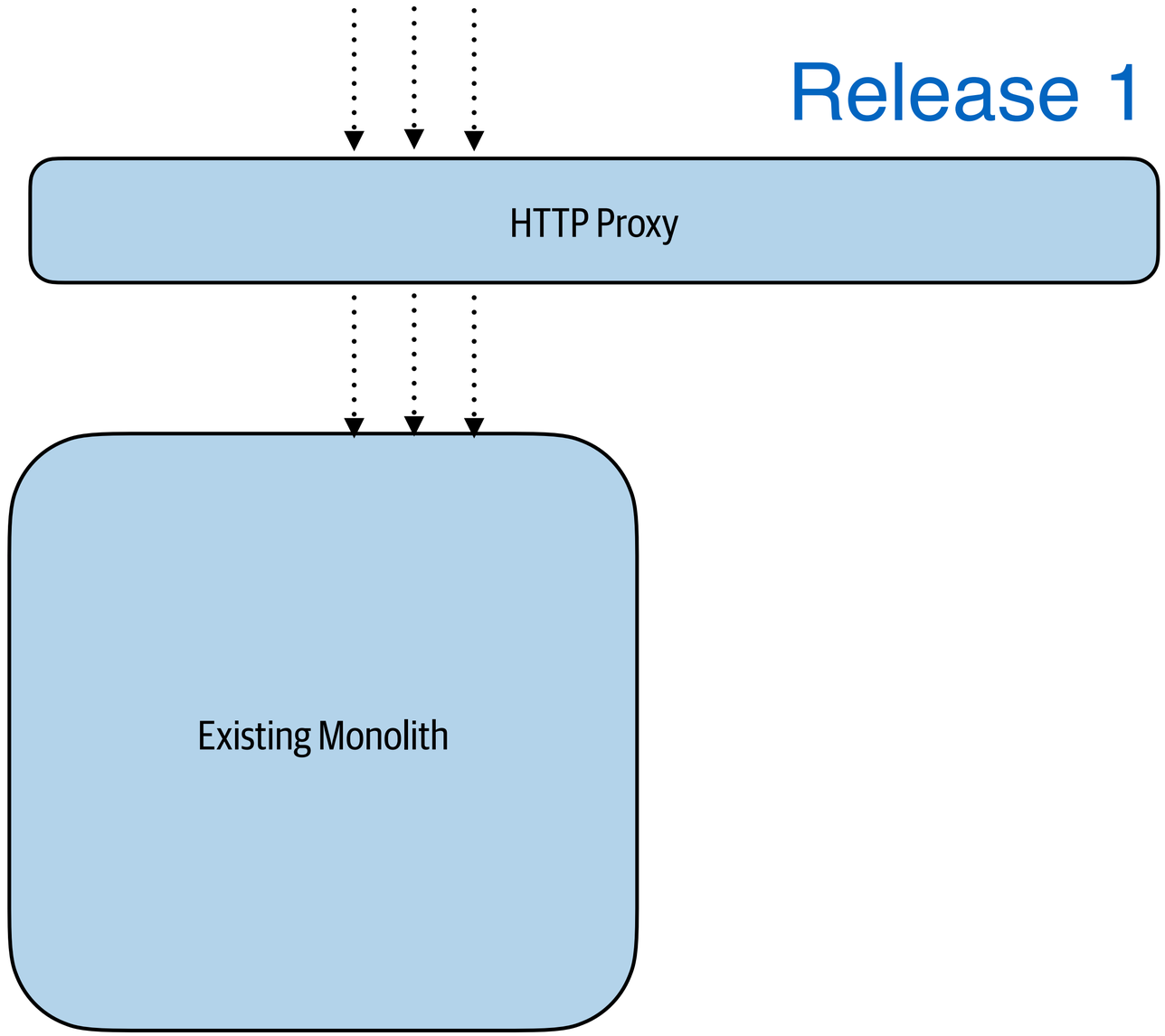


# INCREMENTAL ROLLOUT



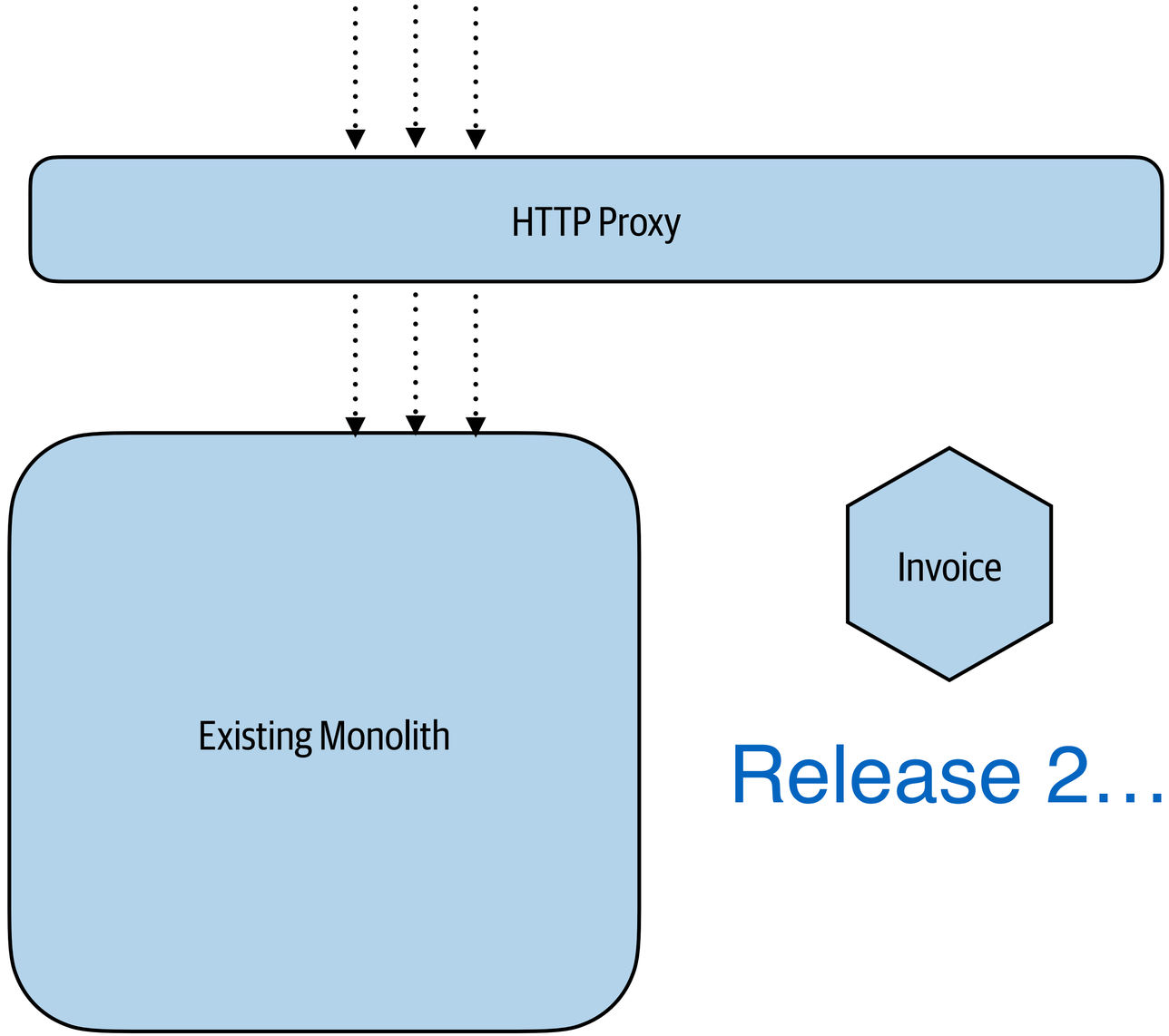
## INCREMENTAL ROLLOUT

Assess impact of adding a network hop early



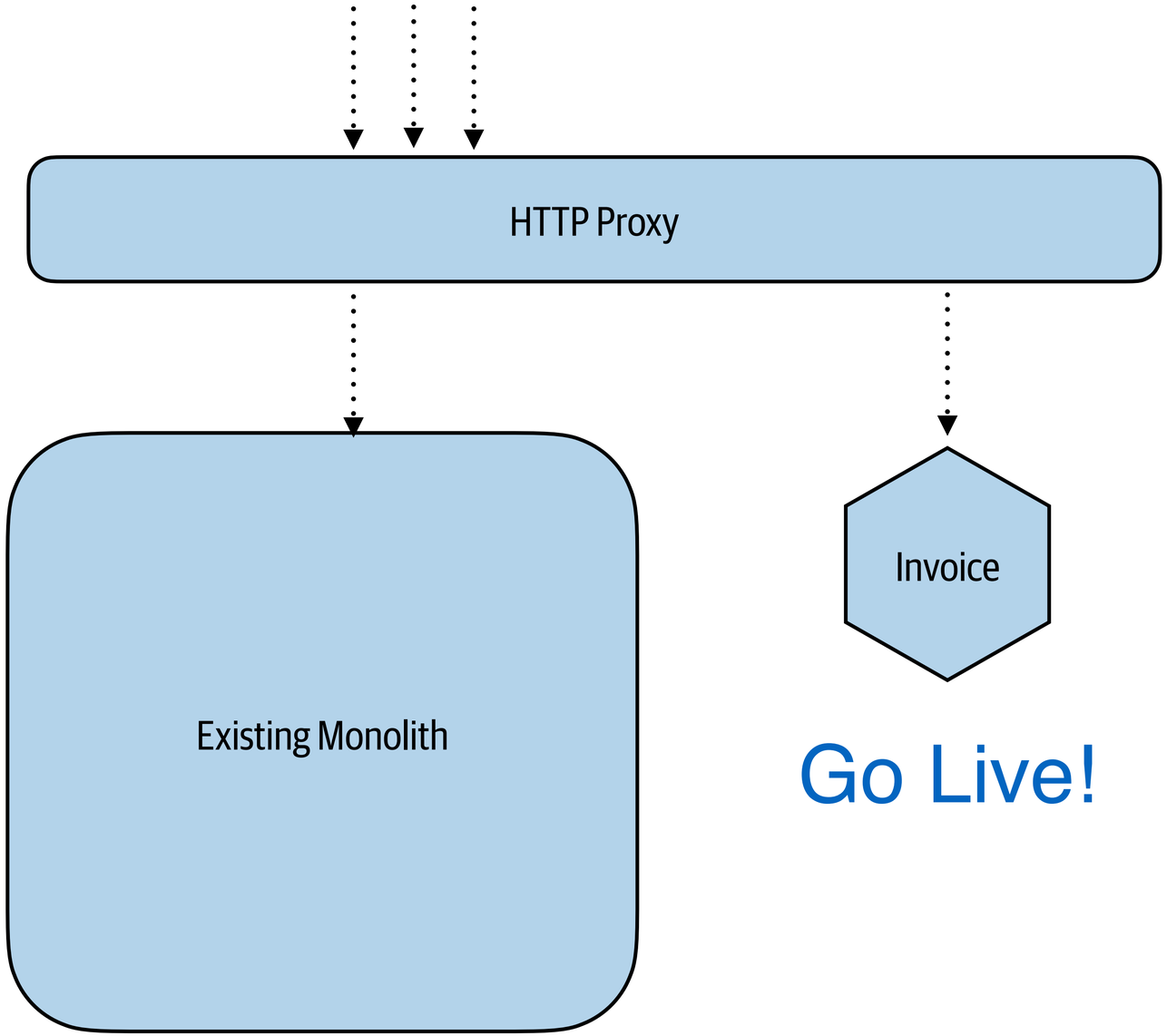
## INCREMENTAL ROLLOUT

Assess impact of adding a network hop early

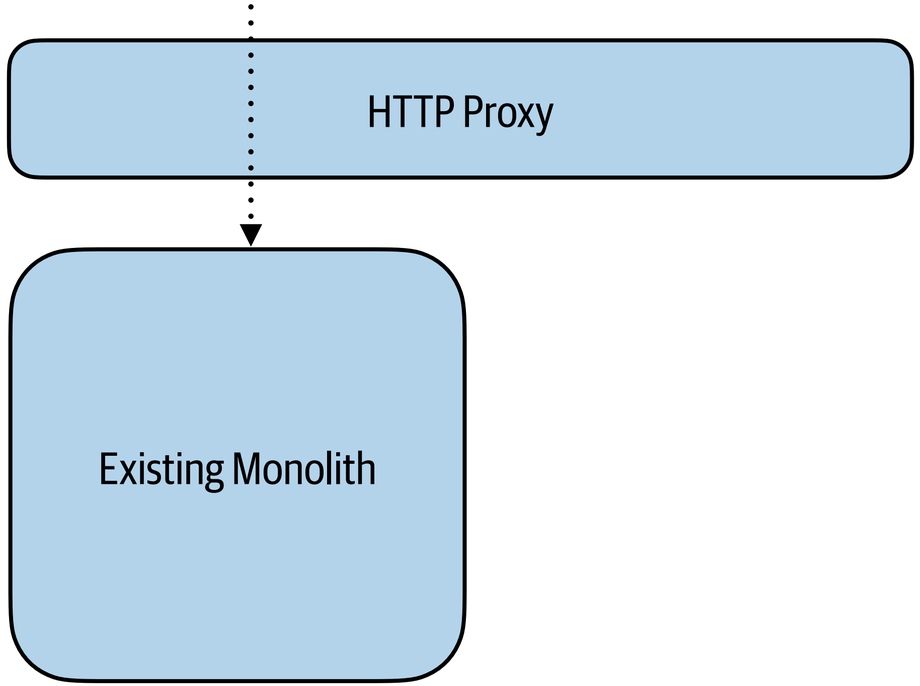


## INCREMENTAL ROLLOUT

Assess impact of adding a network hop early

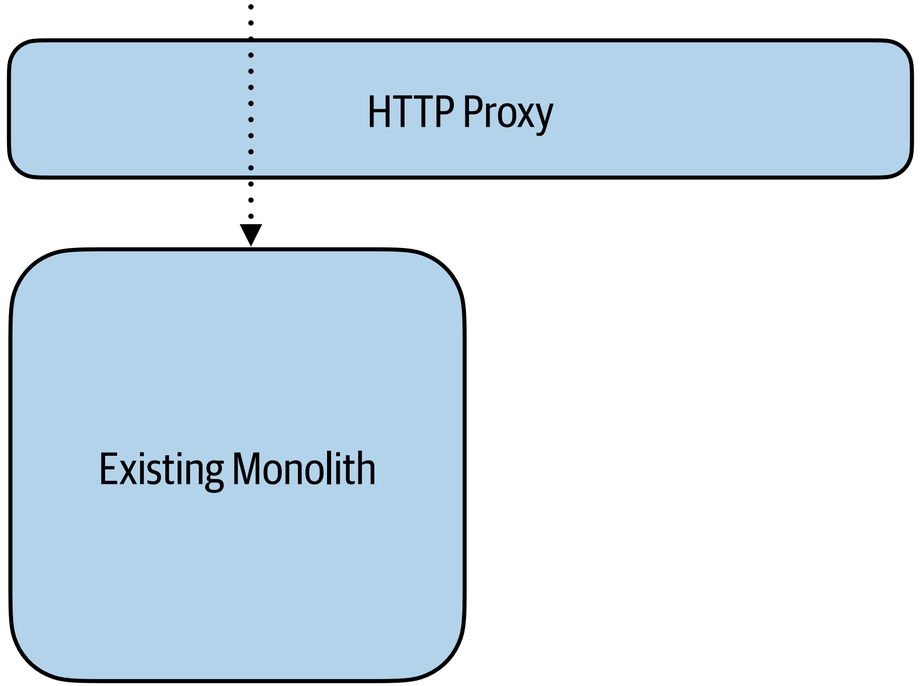


# DEPLOYING WORK THAT ISN'T READY?



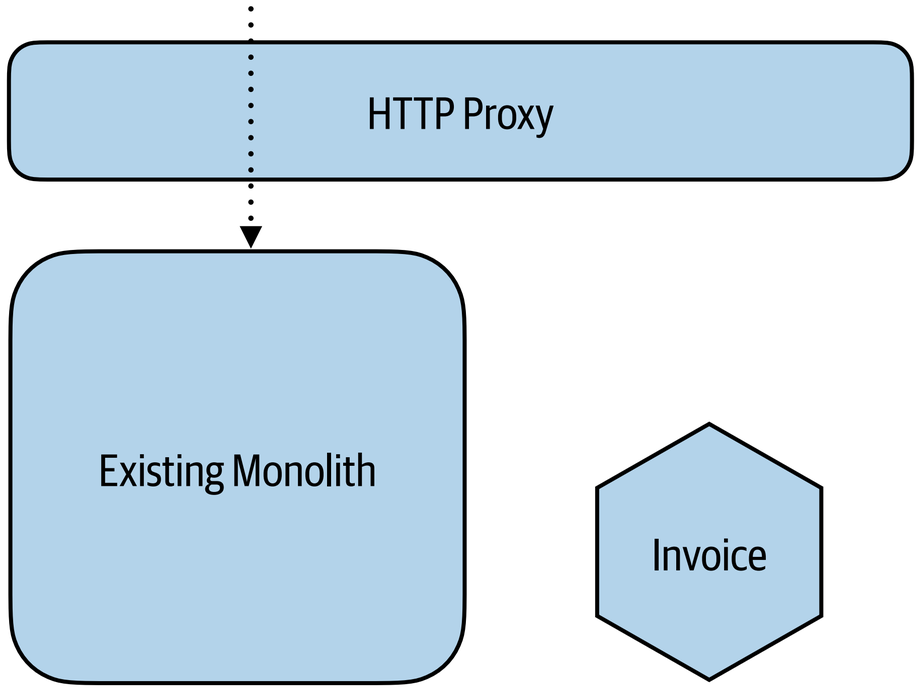
# DEPLOYING WORK THAT ISN'T READY?

Deploying software doesn't need to be conflated with "release"



# DEPLOYING WORK THAT ISN'T READY?

Deploying software doesn't need to be conflated with "release"

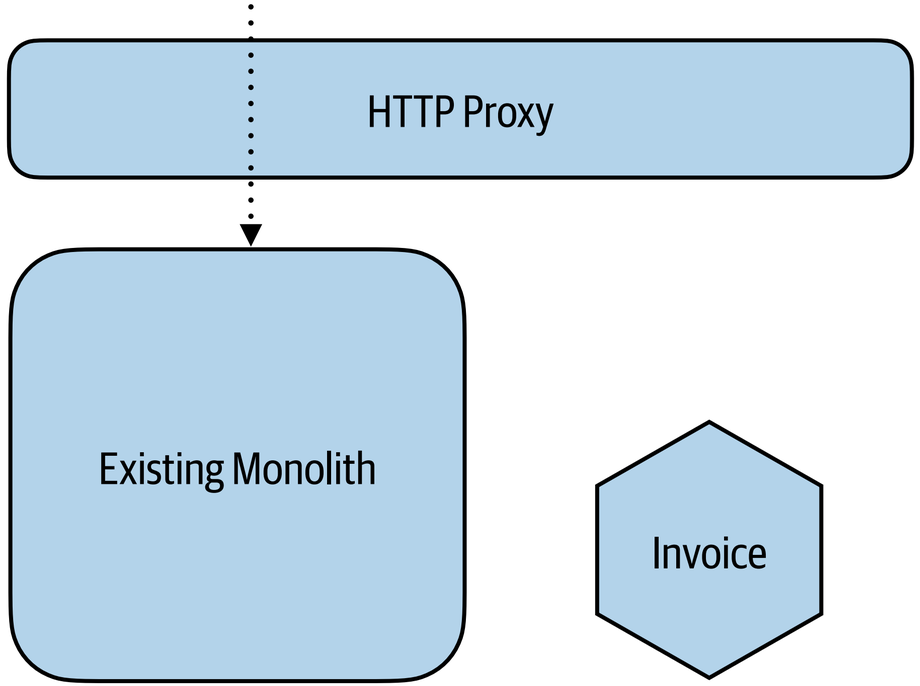



**Invoice microservice is deployed**

# DEPLOYING WORK THAT ISN'T READY?

Deploying software doesn't need to be conflated with "release"

Separating these two steps allows you to do things like validate the newly deployed software prior to release

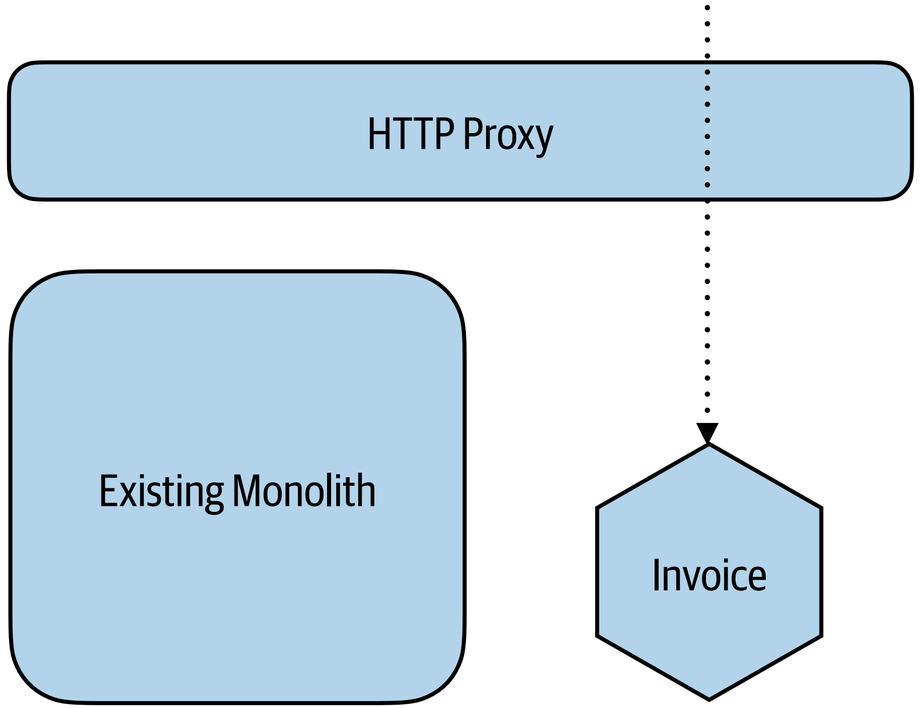



**Invoice microservice is deployed**

# DEPLOYING WORK THAT ISN'T READY?

Deploying software doesn't need to be conflated with "release"

Separating these two steps allows you to do things like validate the newly deployed software prior to release




**Now the new Invoice microservice is released**

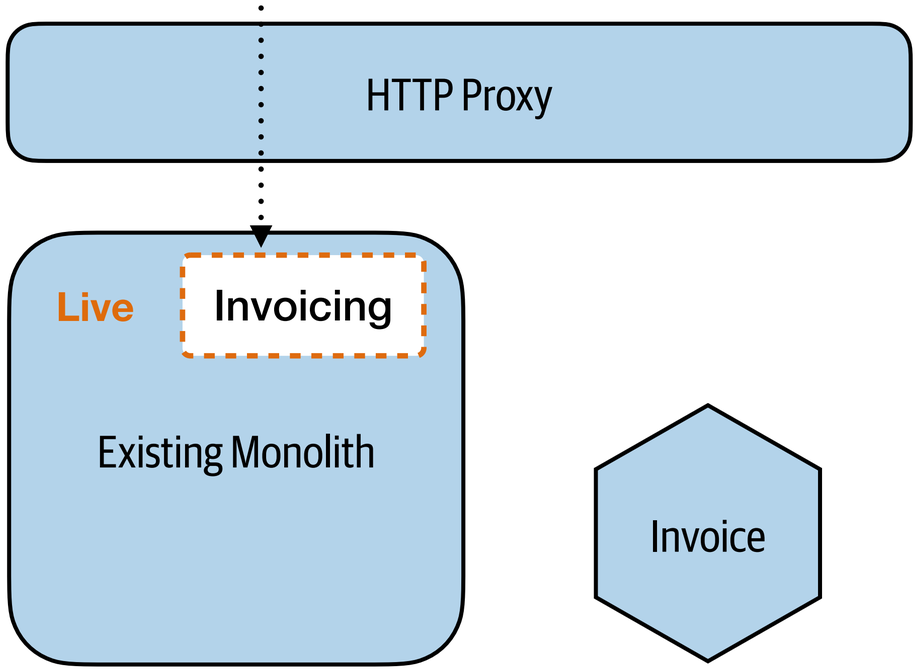
# DON’T REMOVE THE OLD FUNCTIONALITY TOO SOON!

## DON’T REMOVE THE OLD FUNCTIONALITY TOO SOON!

If we leave the old invoicing functionality in the monolith, we have an easy rollback

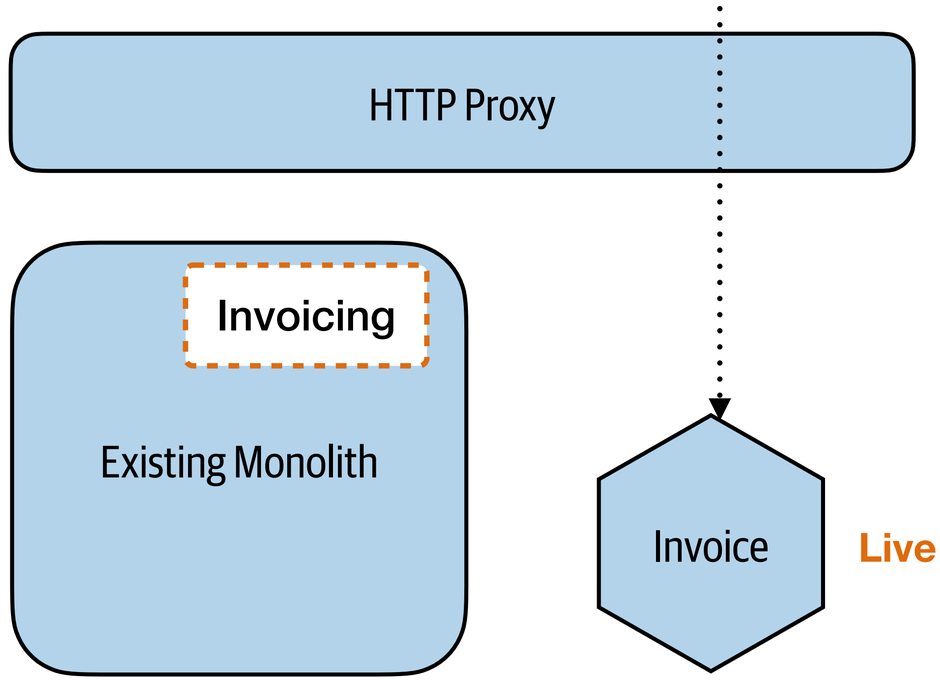
# DON’T REMOVE THE OLD FUNCTIONALITY TOO SOON!

If we leave the old invoicing functionality in the monolith, we have an easy rollback



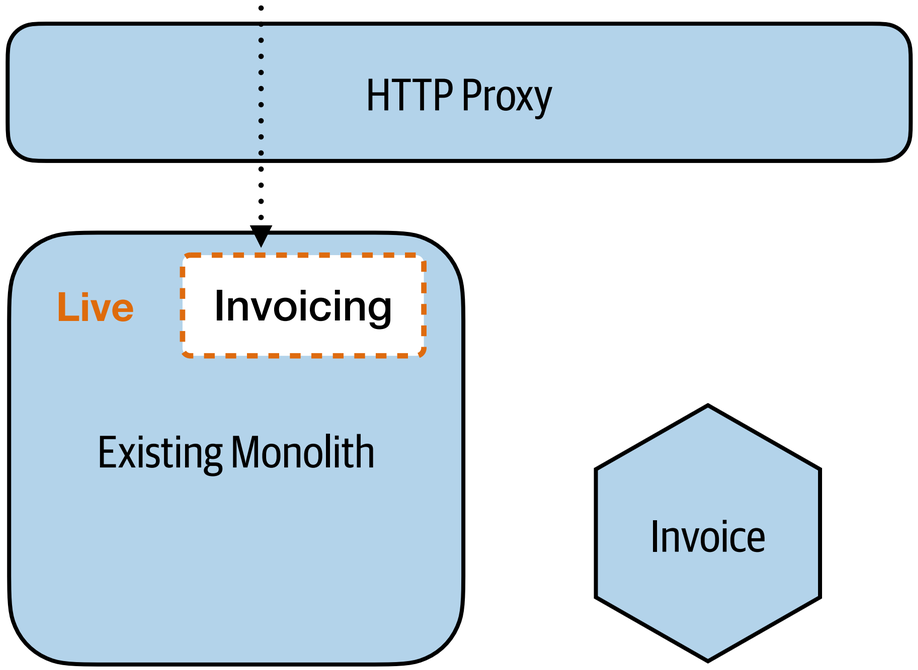
# DON’T REMOVE THE OLD FUNCTIONALITY TOO SOON!

If we leave the old invoicing functionality in the monolith, we have an easy rollback



# DON’T REMOVE THE OLD FUNCTIONALITY TOO SOON!

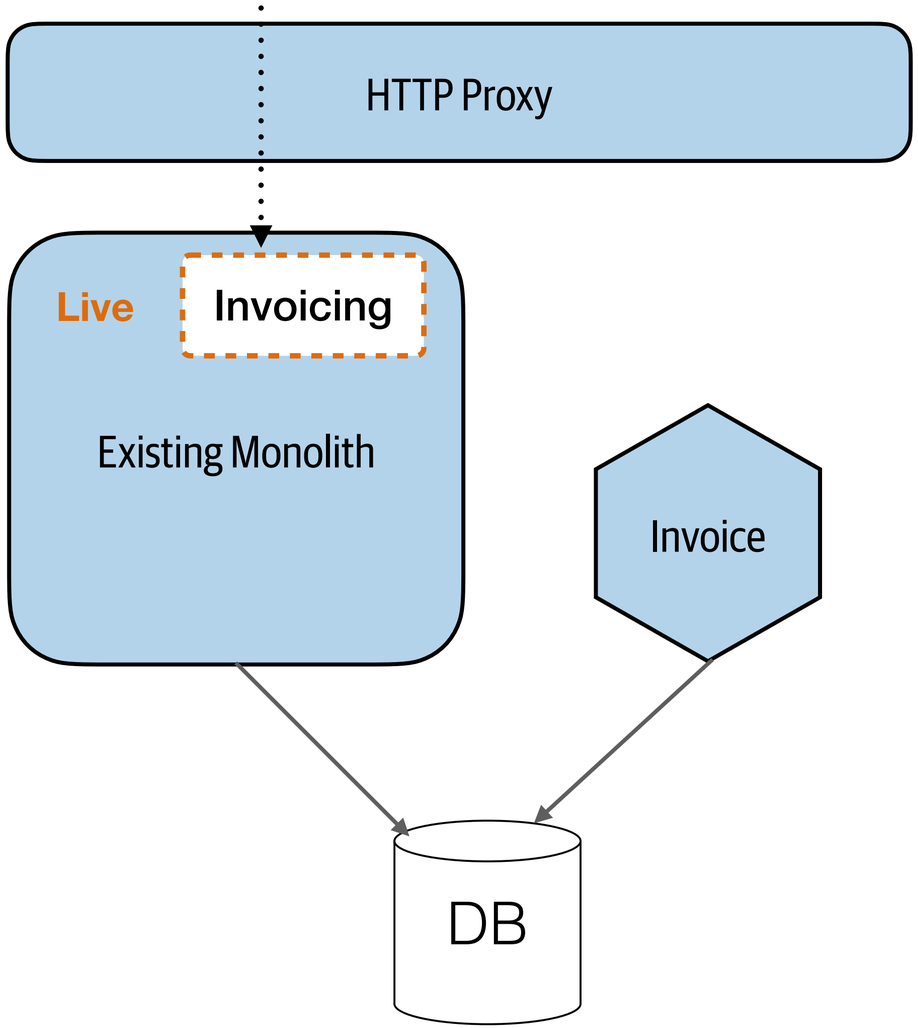
If we leave the old invoicing functionality in the monolith, we have an easy rollback



# DON’T REMOVE THE OLD FUNCTIONALITY TOO SOON!

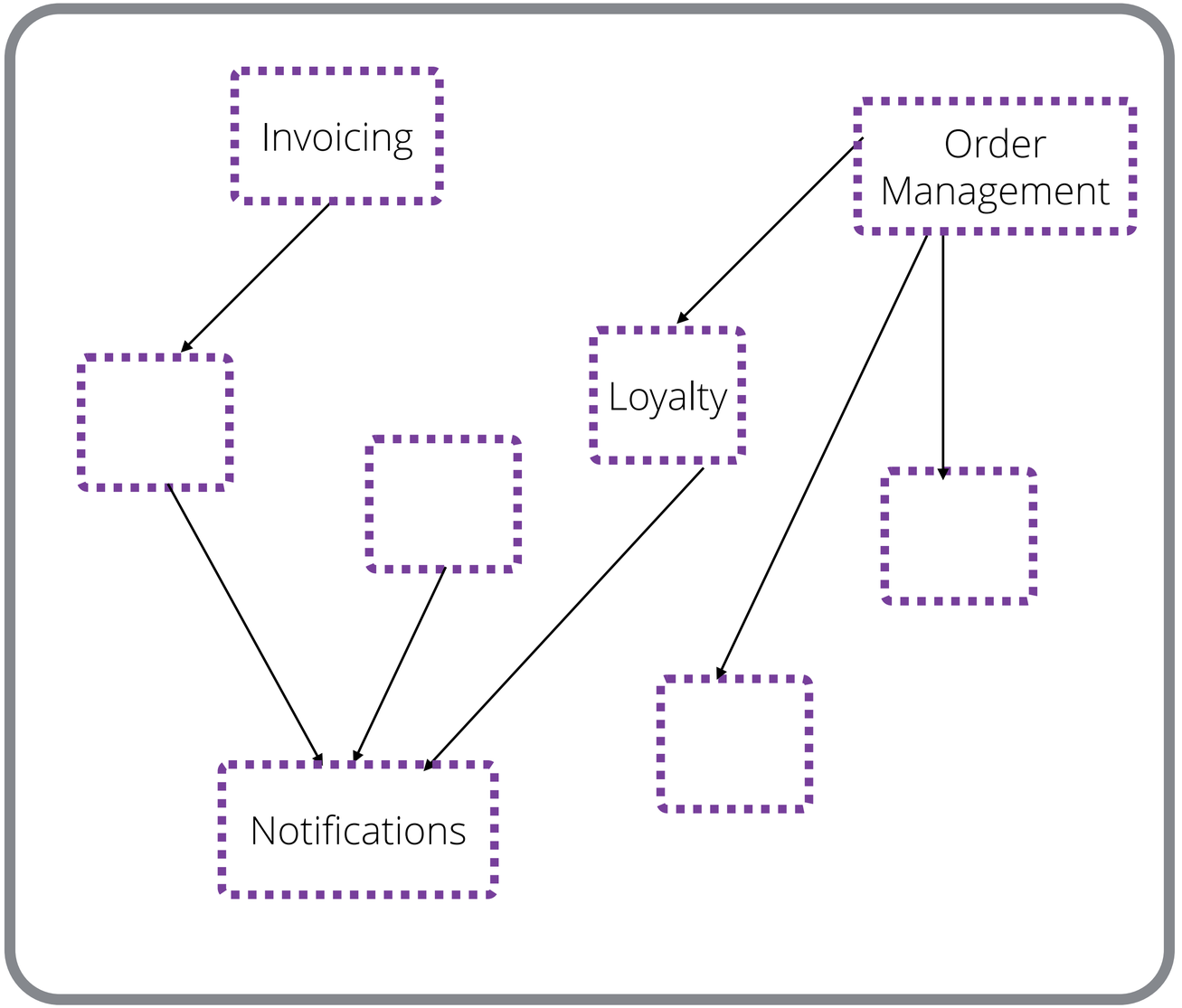
If we leave the old invoicing functionality in the monolith, we have an easy rollback

Would likely need to use the same DB during this transition



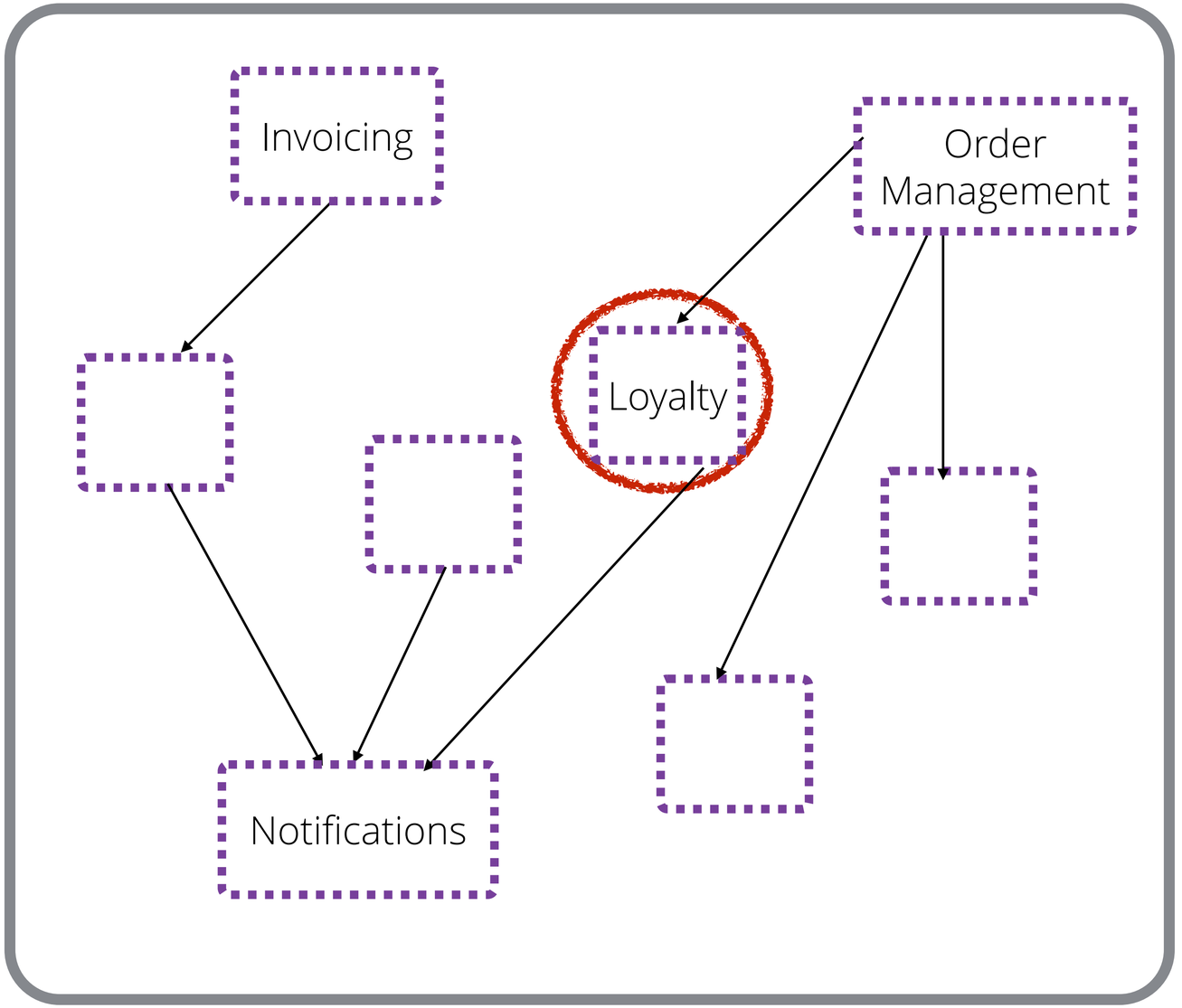
# WHAT ABOUT OTHER FUNCTIONALITY?

Can use “branch by abstraction” to help extraction of functionality deeper in the monolith



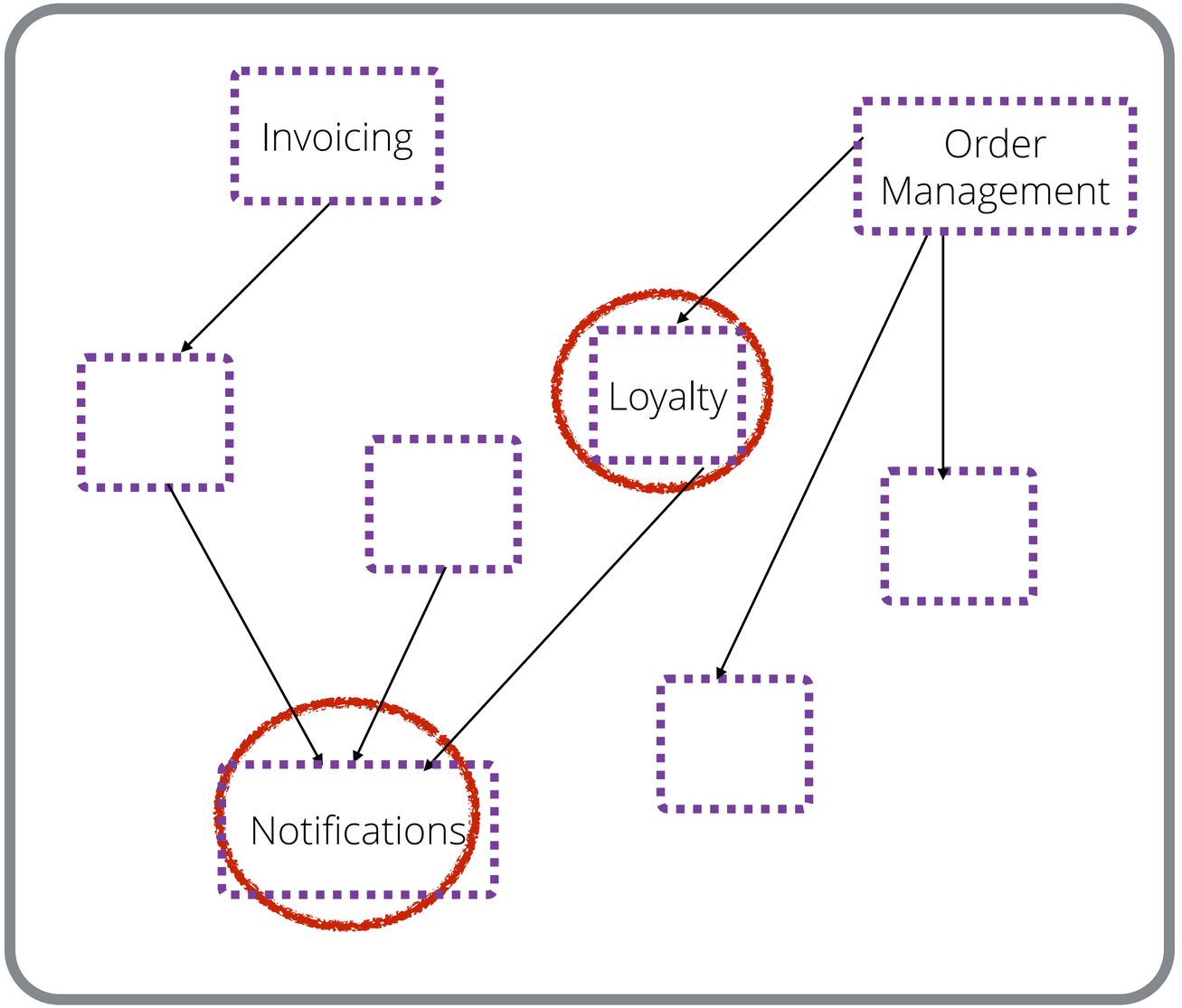
# WHAT ABOUT OTHER FUNCTIONALITY?

Can use “branch by abstraction” to help extraction of functionality deeper in the monolith

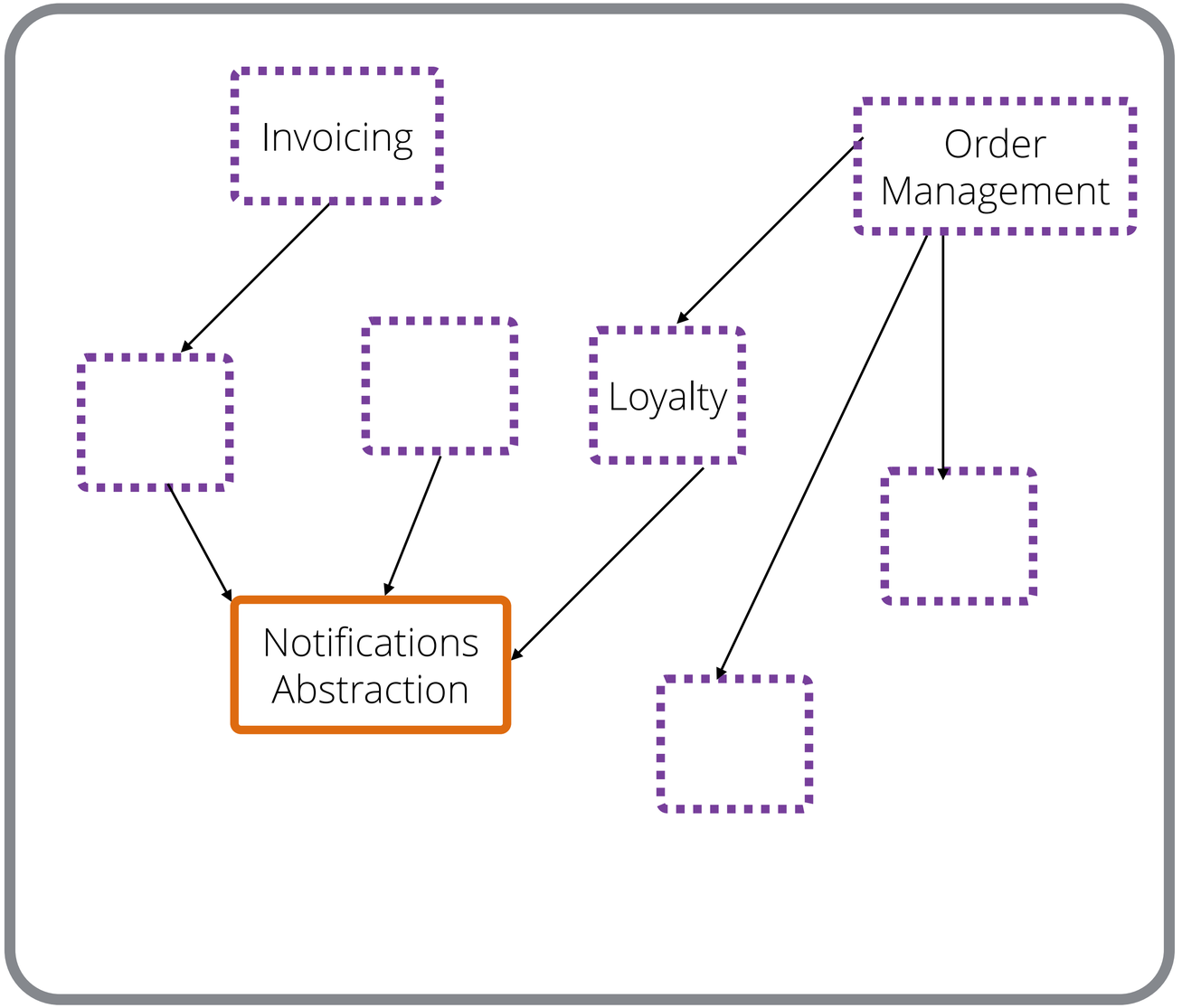


# WHAT ABOUT OTHER FUNCTIONALITY?

Can use “branch by abstraction” to help extraction of functionality deeper in the monolith

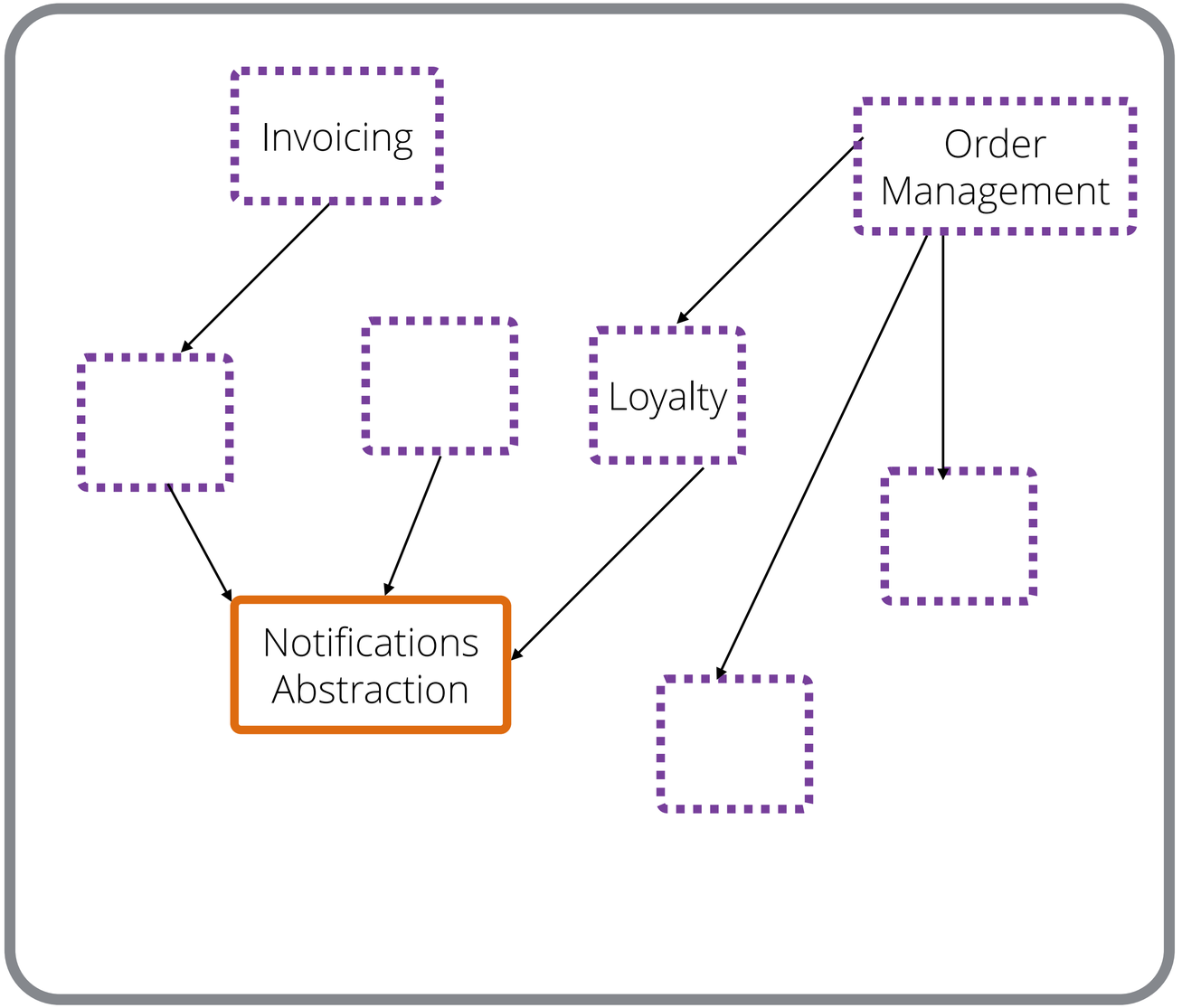


# BRANCH BY ABSTRACTION



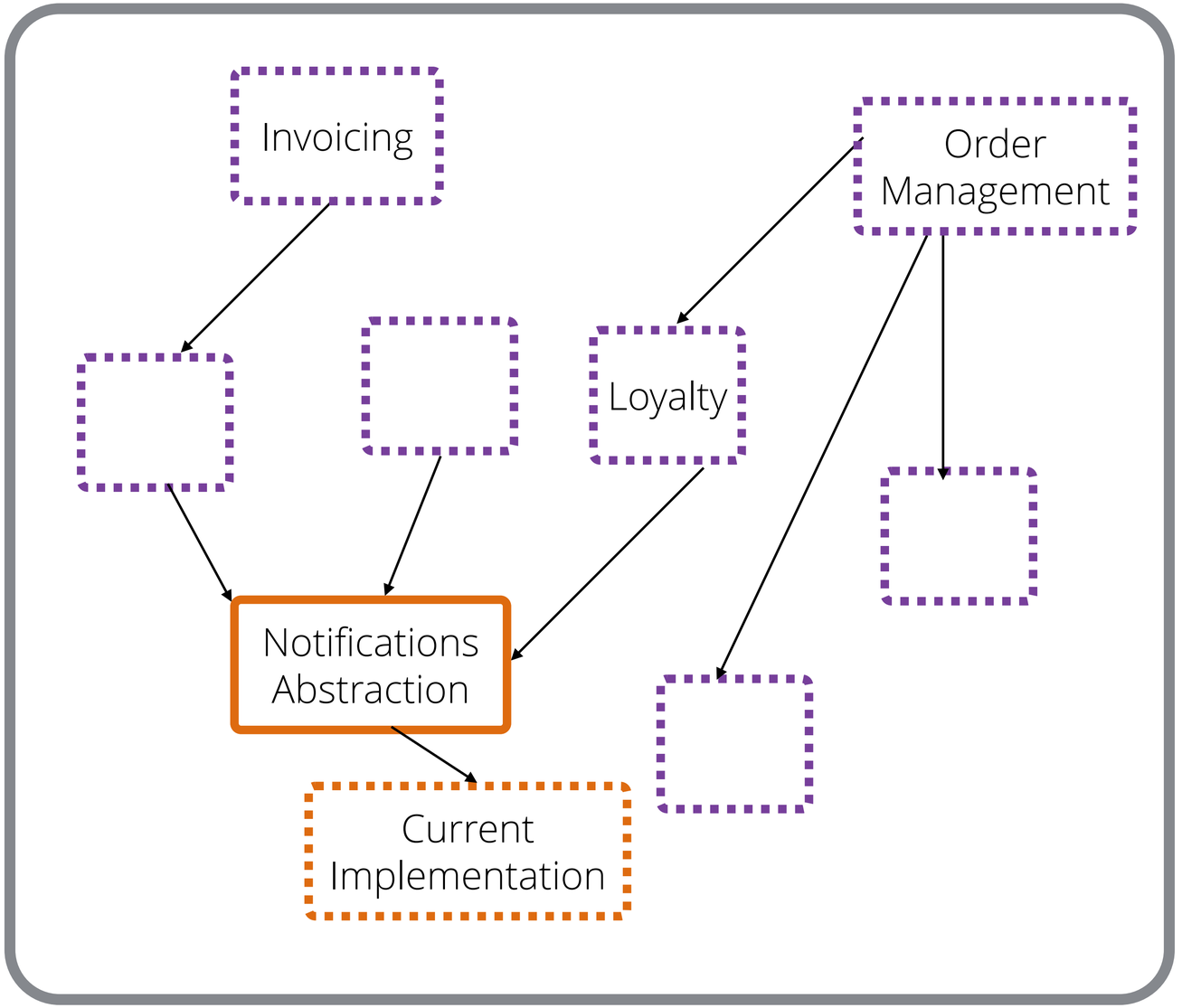
# BRANCH BY ABSTRACTION

**Gives you an abstraction point which can be used to “hide” ongoing development**



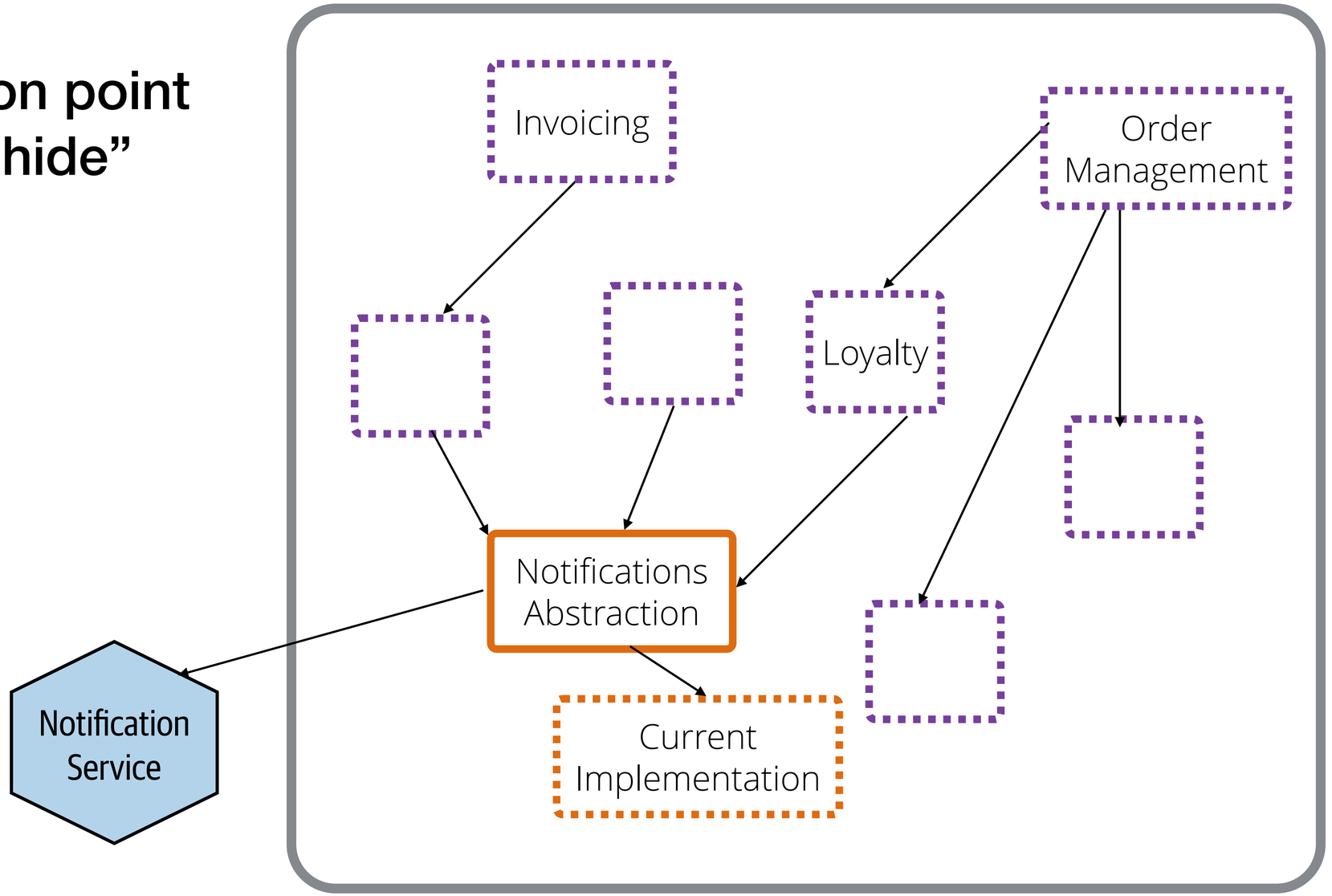
# BRANCH BY ABSTRACTION

**Gives you an abstraction point which can be used to “hide” ongoing development**



# BRANCH BY ABSTRACTION

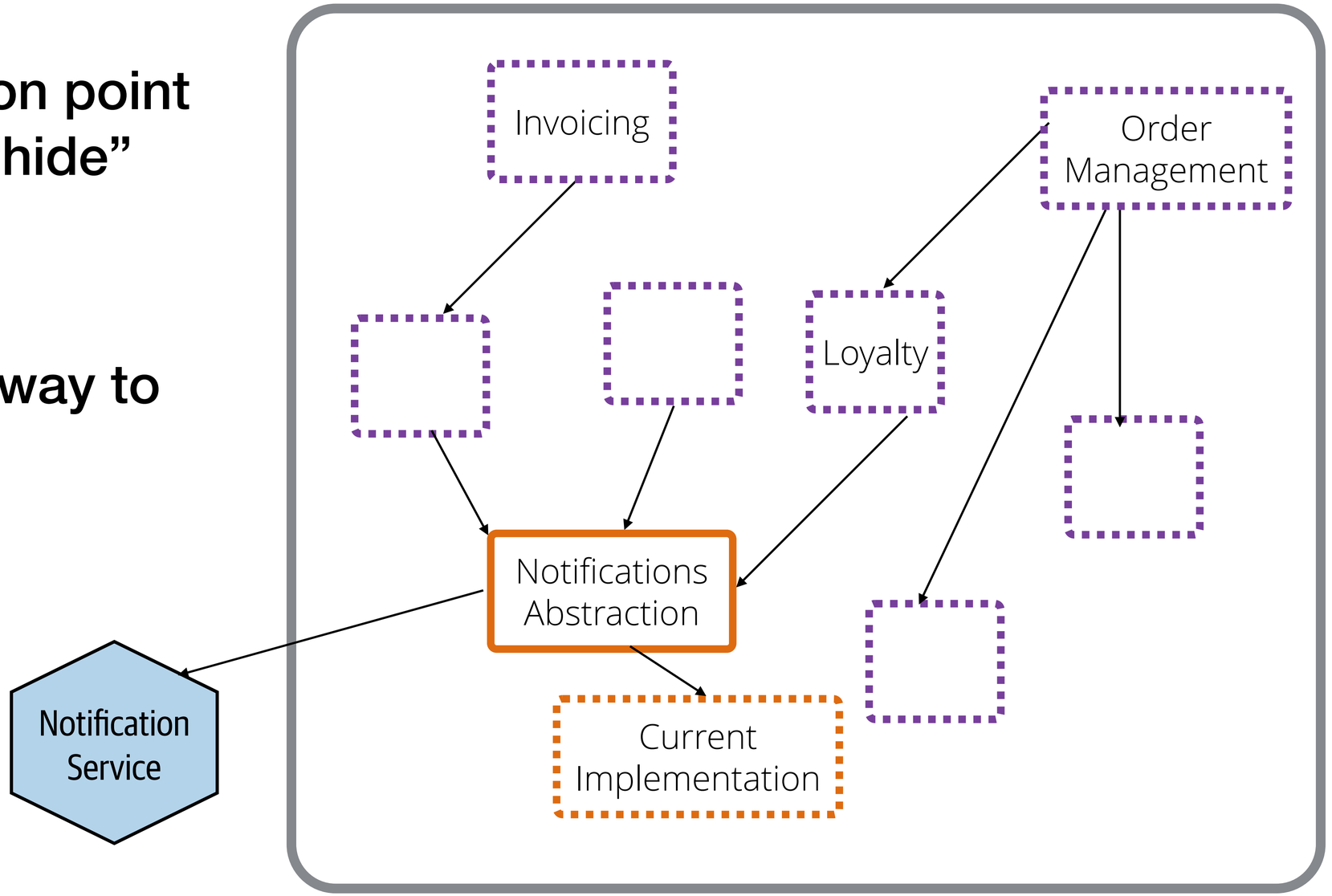
**Gives you an abstraction point which can be used to “hide” ongoing development**



# BRANCH BY ABSTRACTION

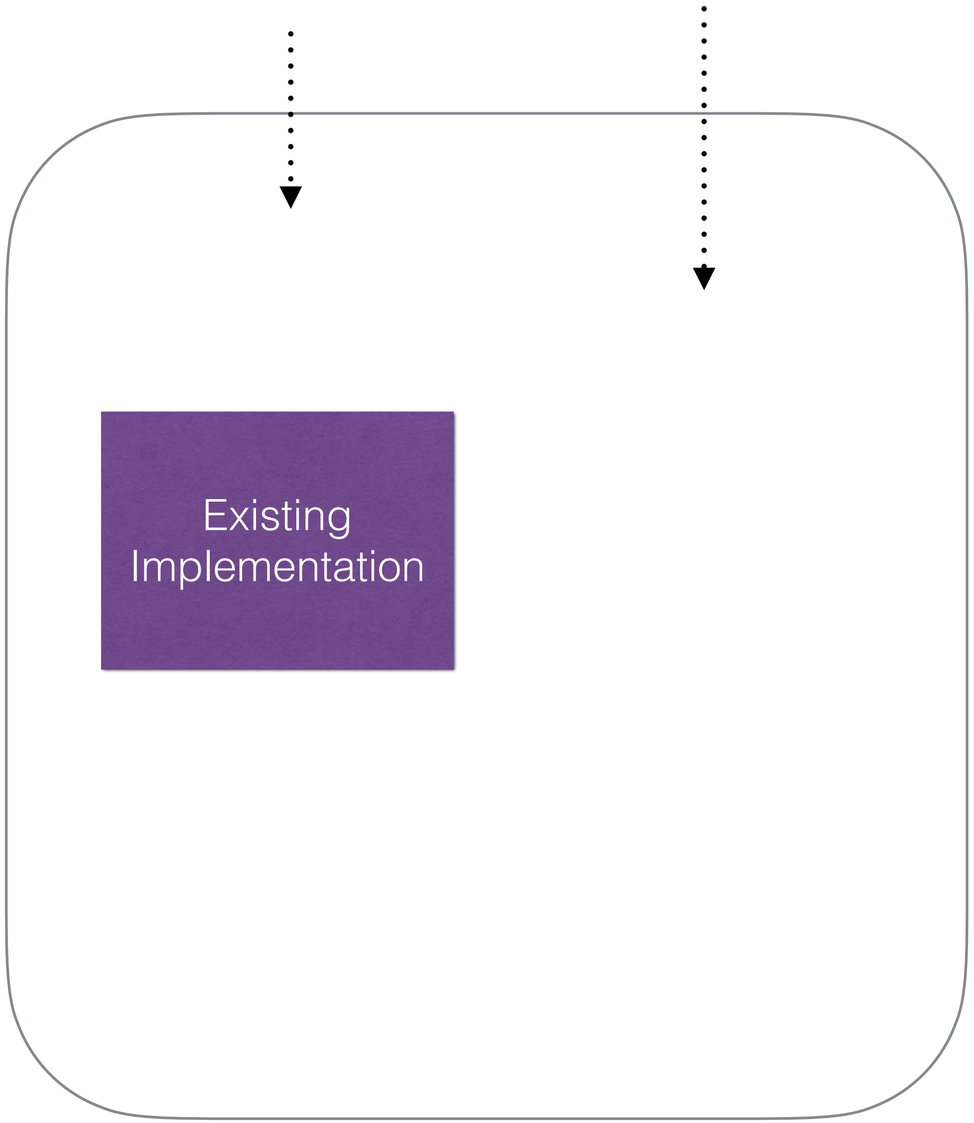
Gives you an abstraction point which can be used to “hide” ongoing development

Can also be used as a way to toggle between implementations

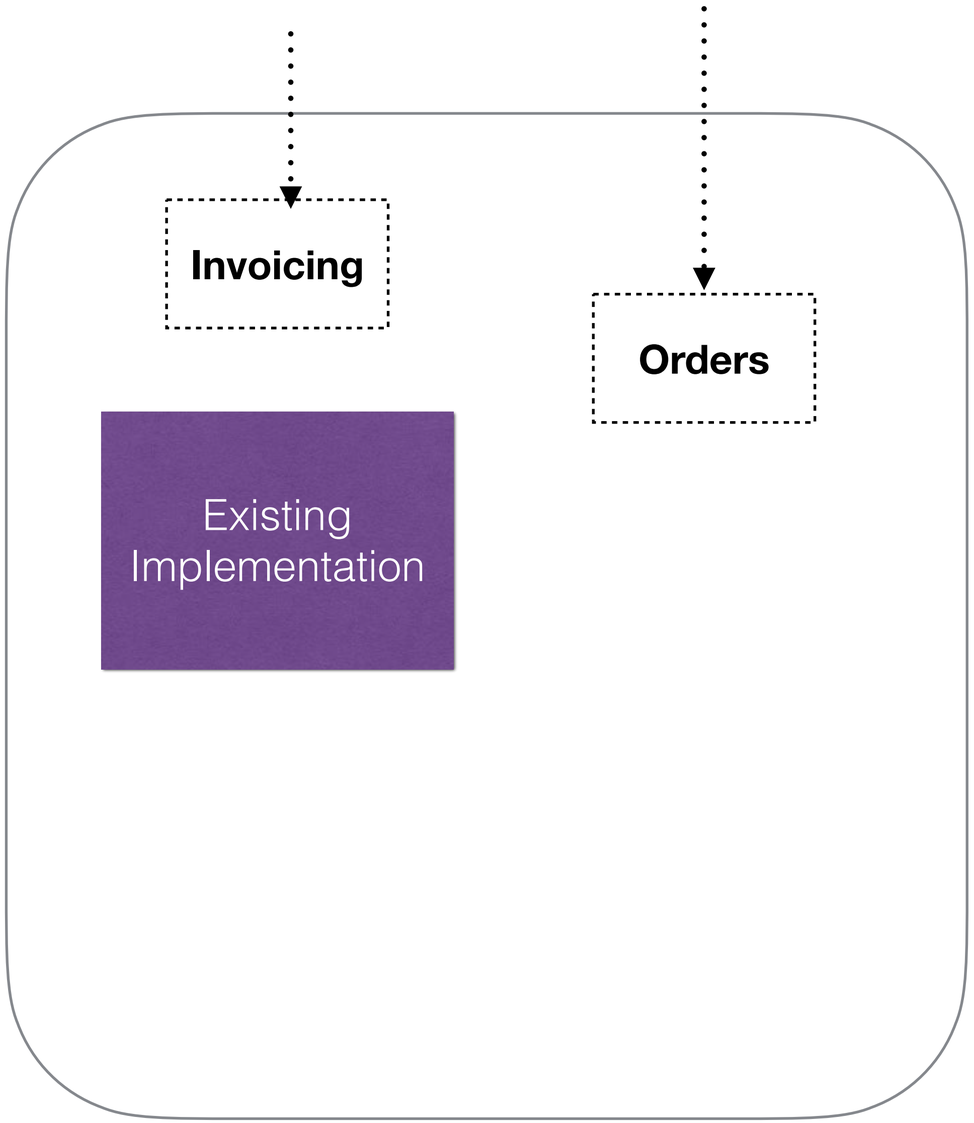


# EXAMPLE

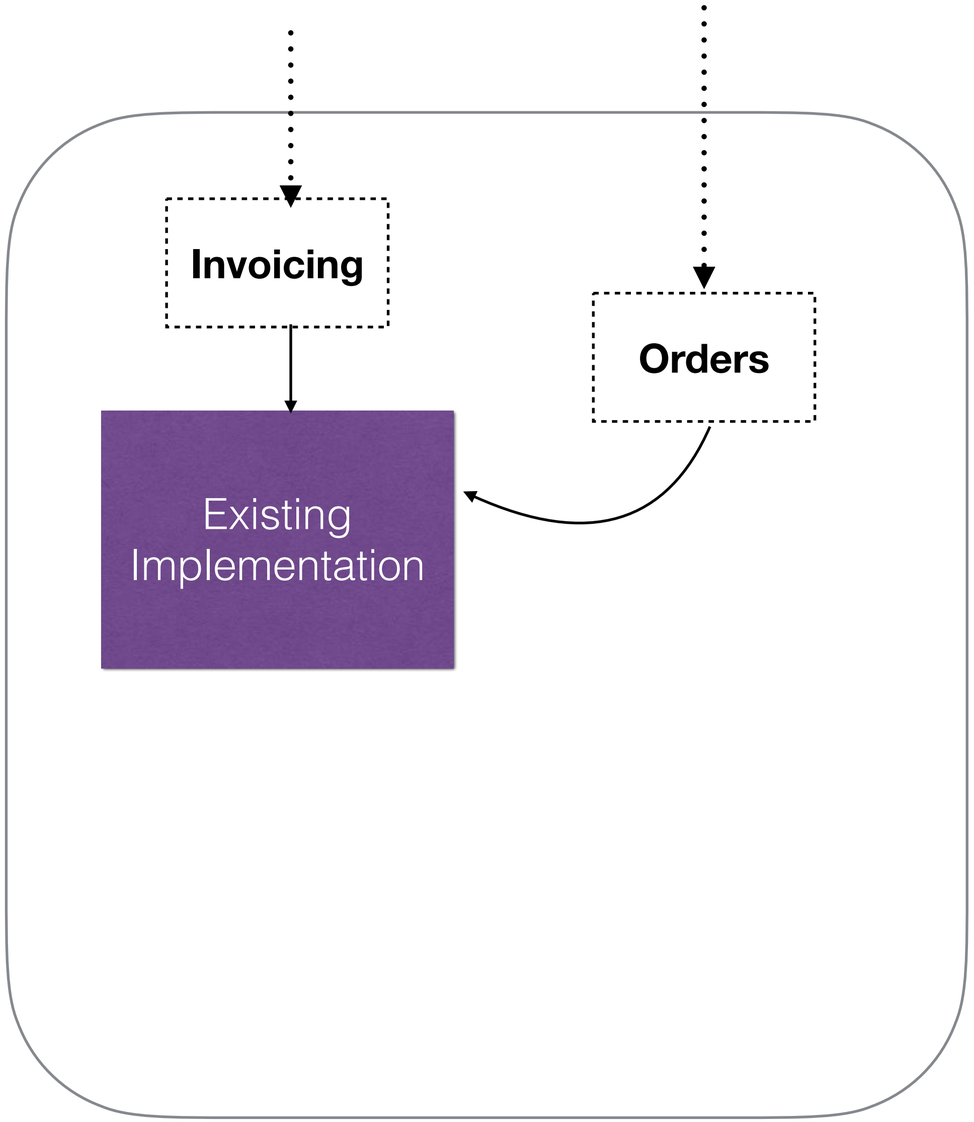
# EXAMPLE



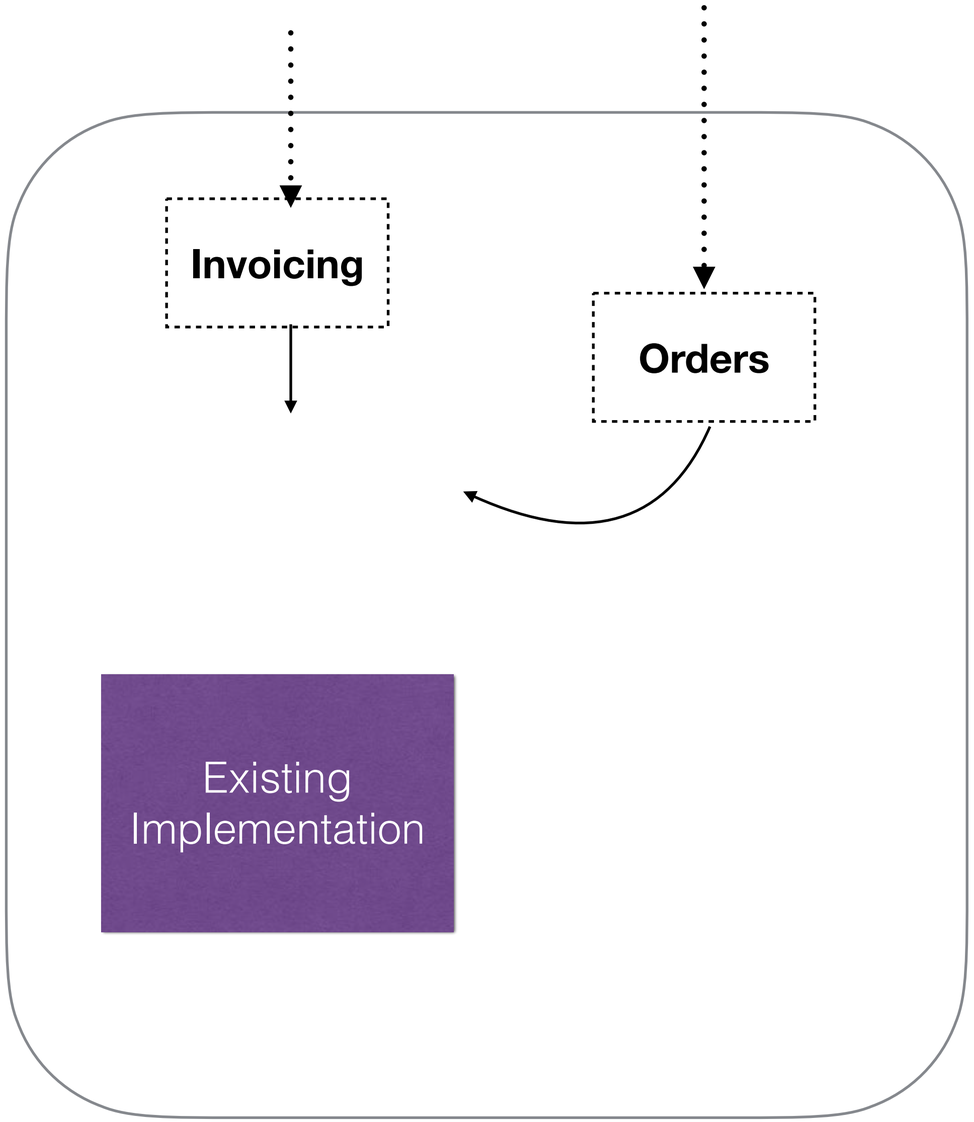
# EXAMPLE



# EXAMPLE

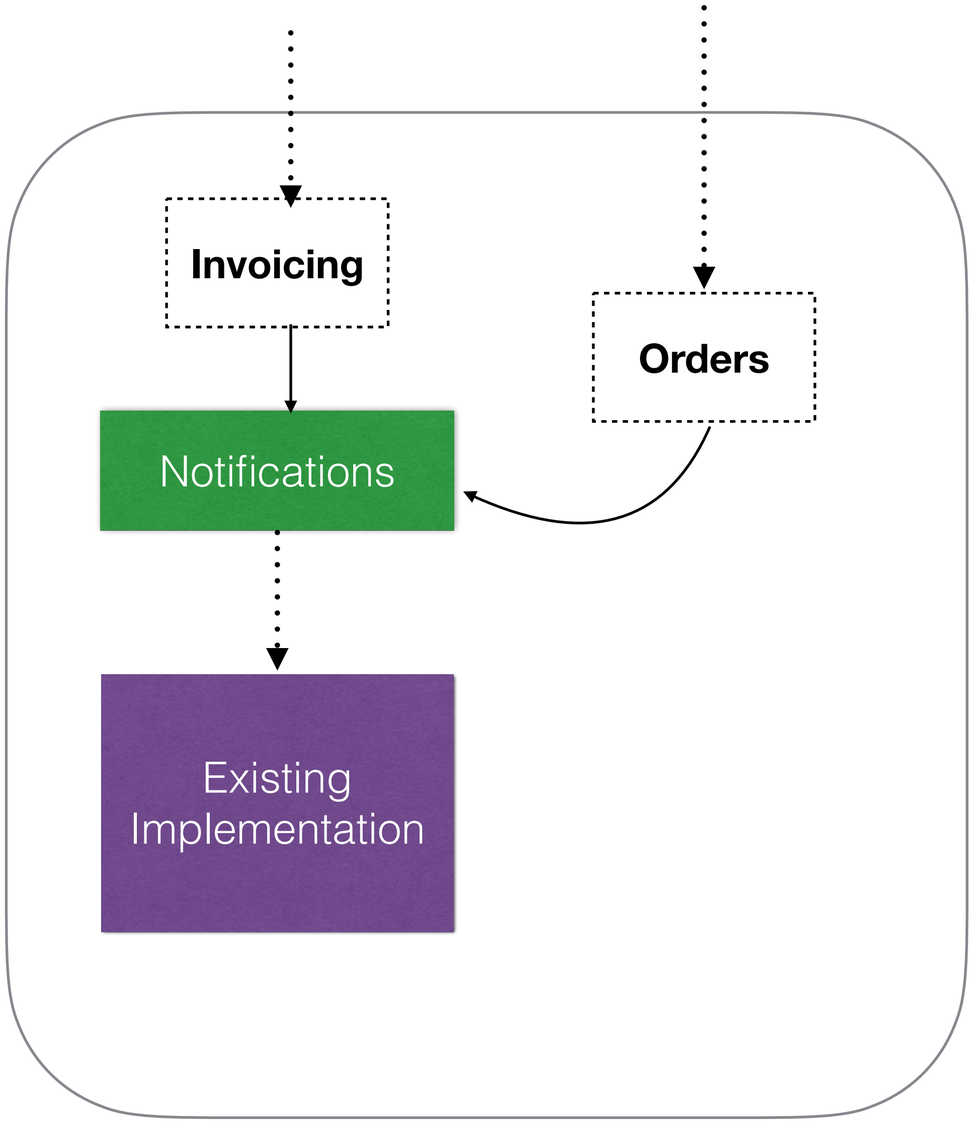


# EXAMPLE

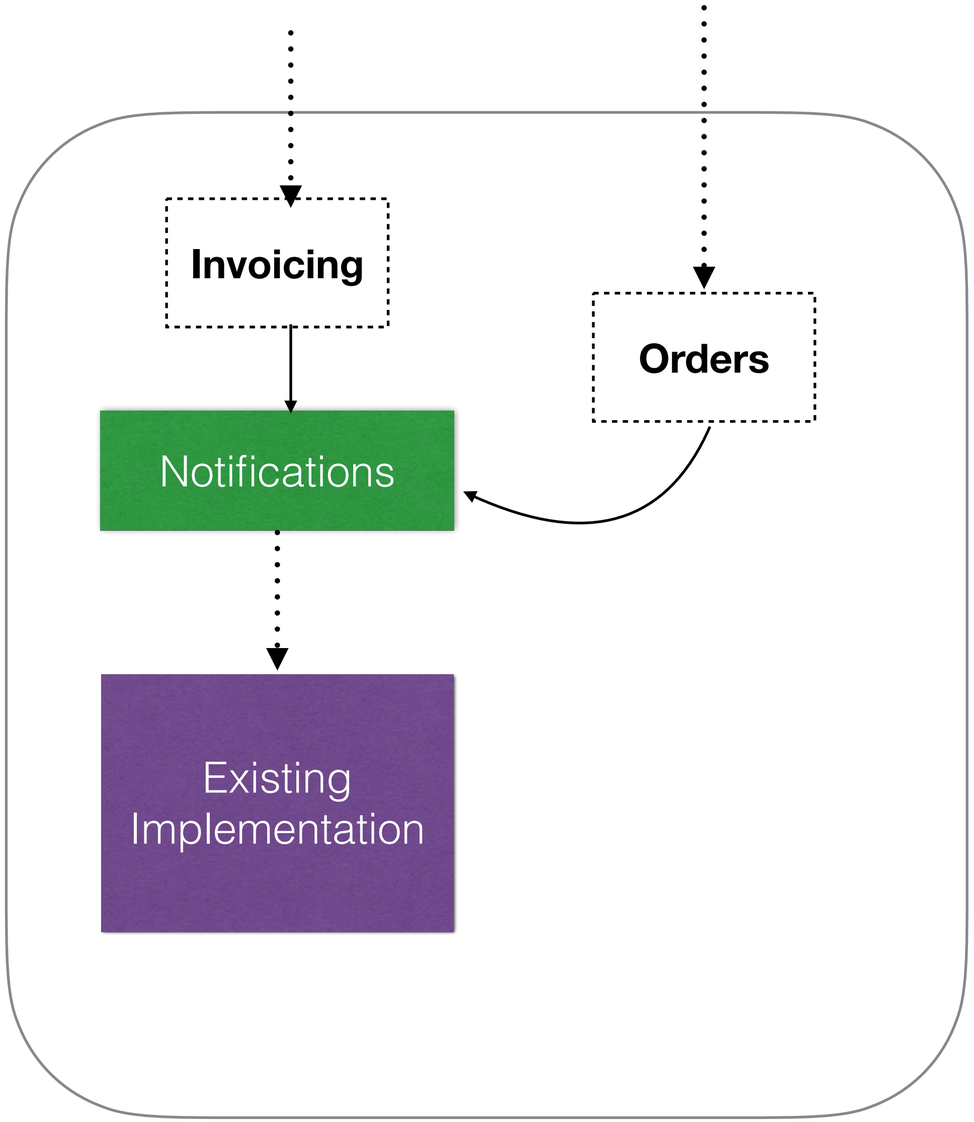


## EXAMPLE

# 1. Create abstraction point



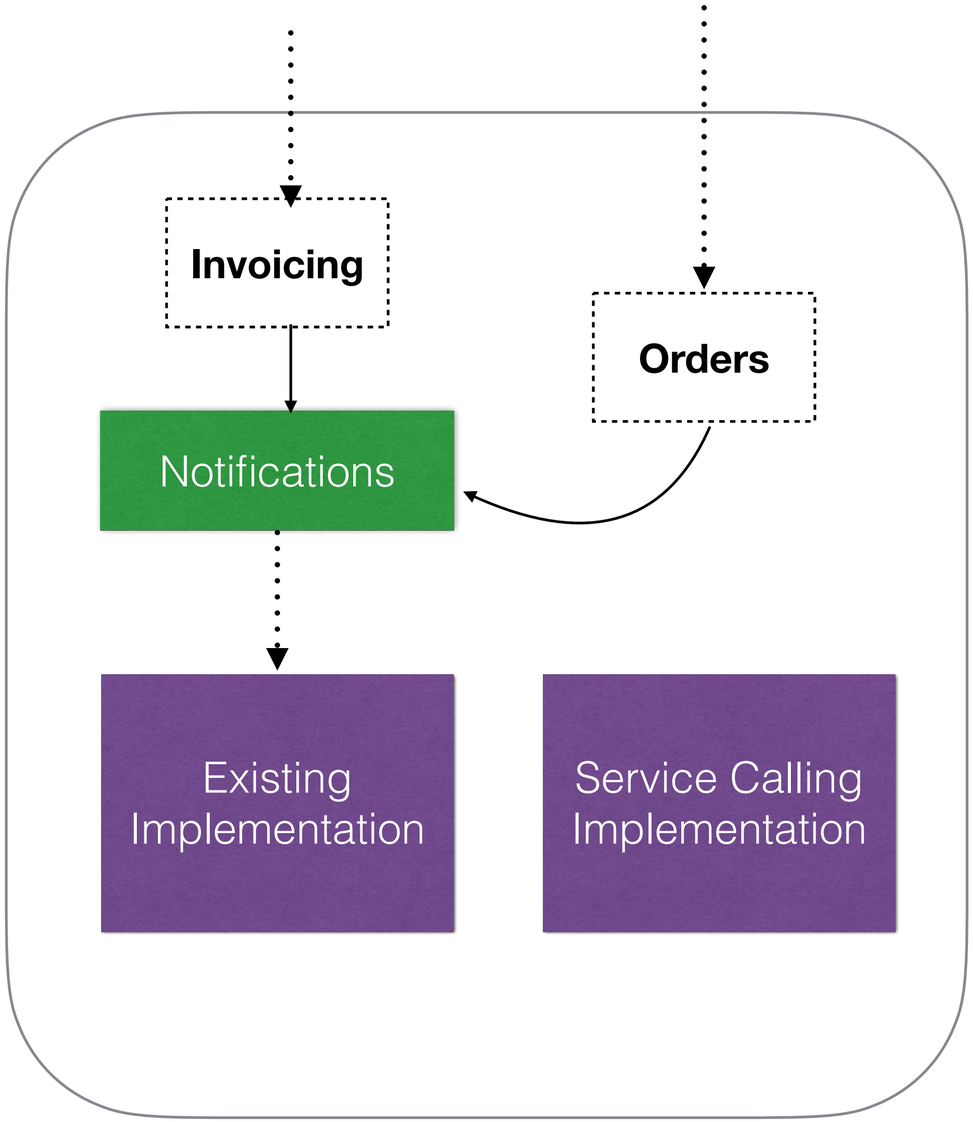
# EXAMPLE




1. Create abstraction point

2. Start work on new service implementation

# EXAMPLE



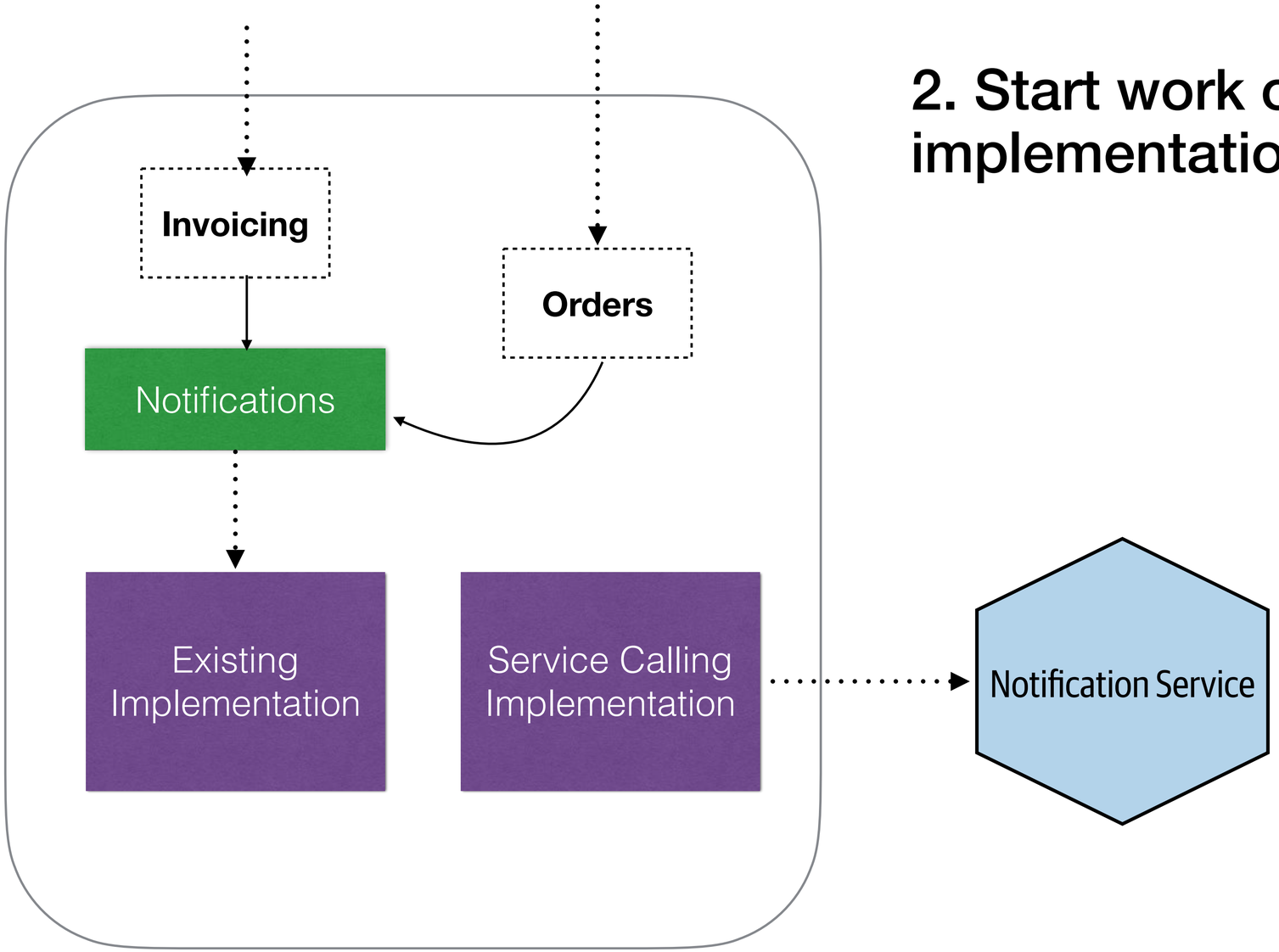

1. Create abstraction point

2. Start work on new service implementation

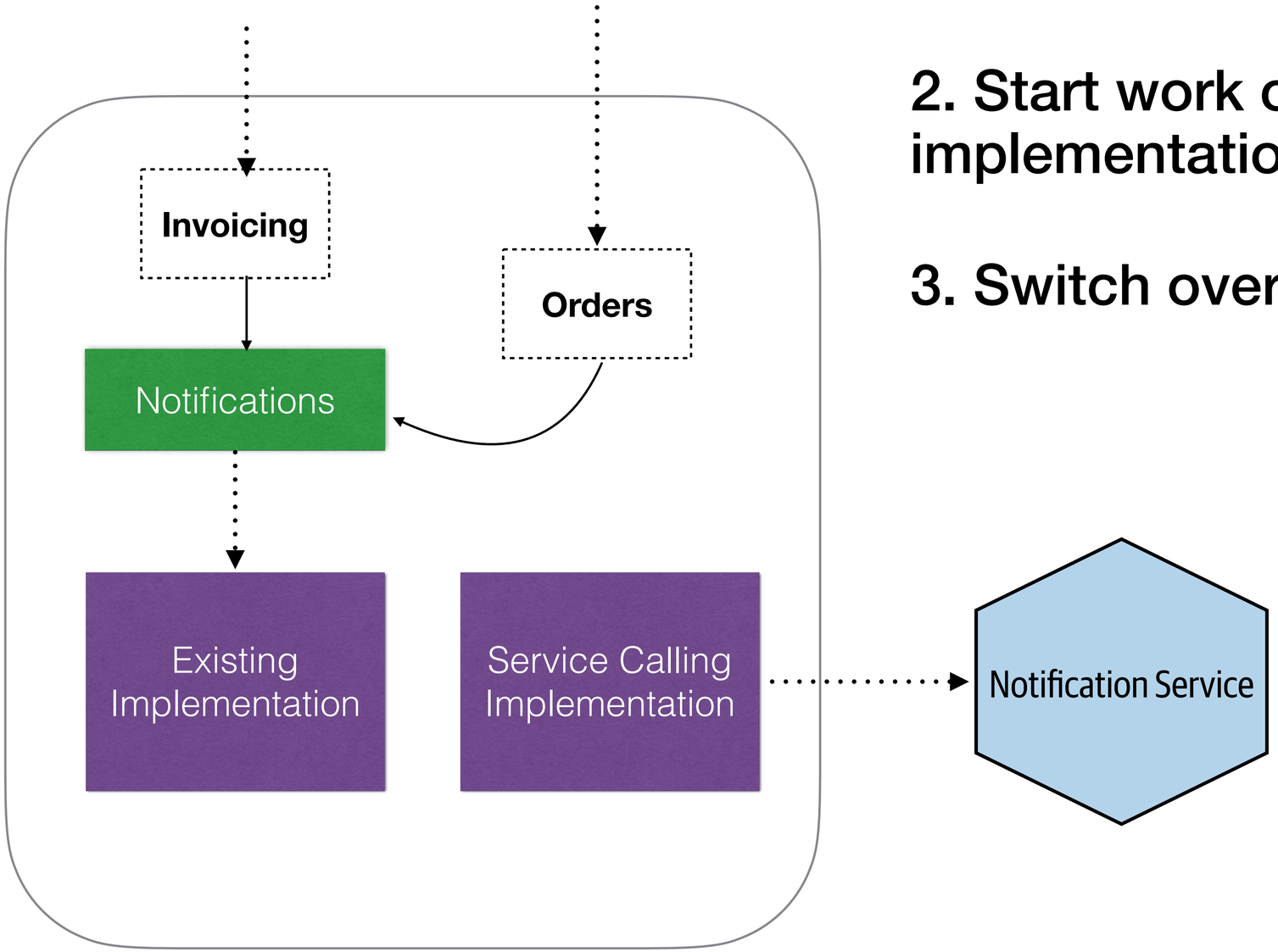
# EXAMPLE

1. Create abstraction point

2. Start work on new service implementation



# EXAMPLE

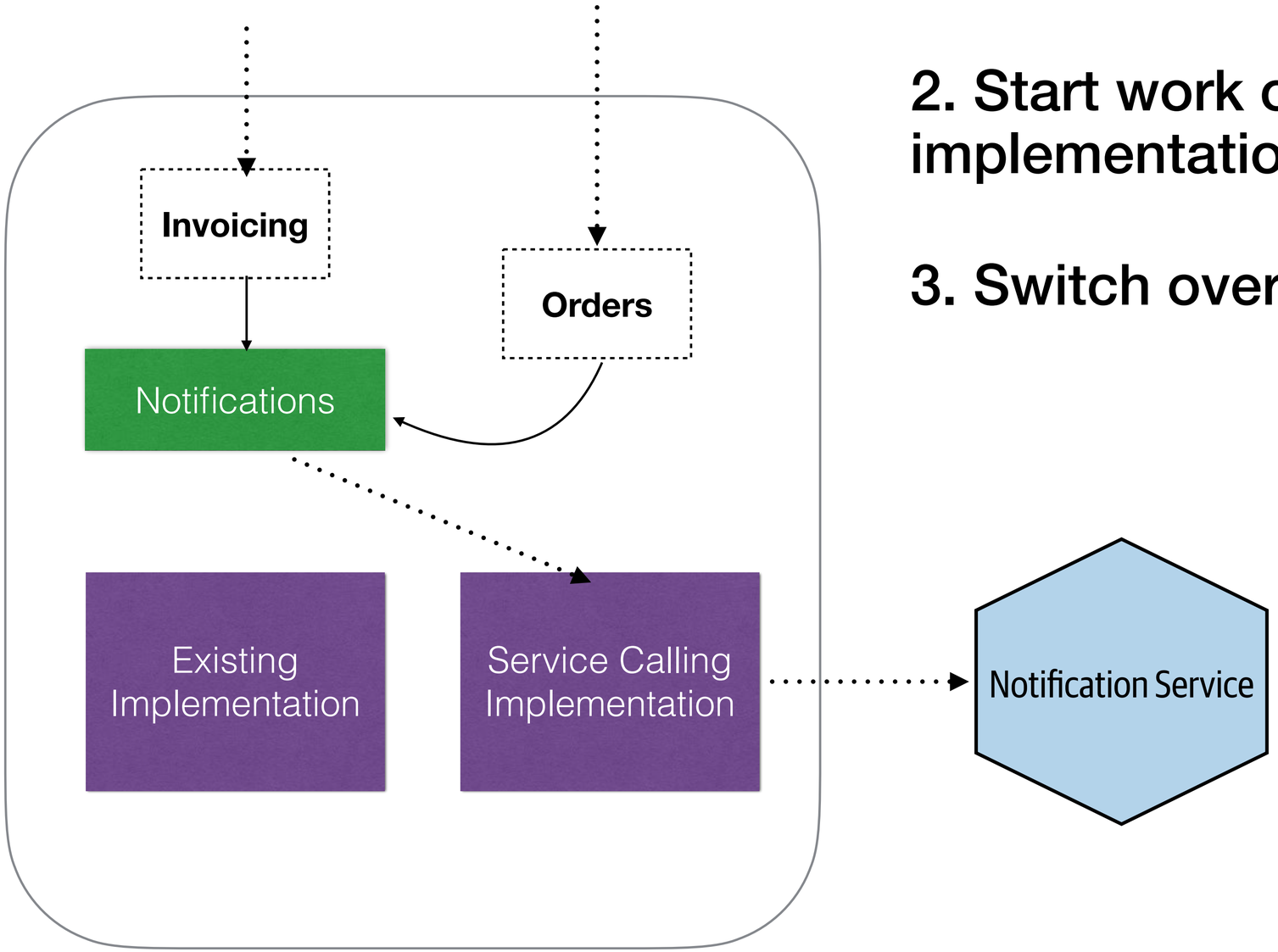



1. Create abstraction point

2. Start work on new service implementation

3. Switch over

# EXAMPLE

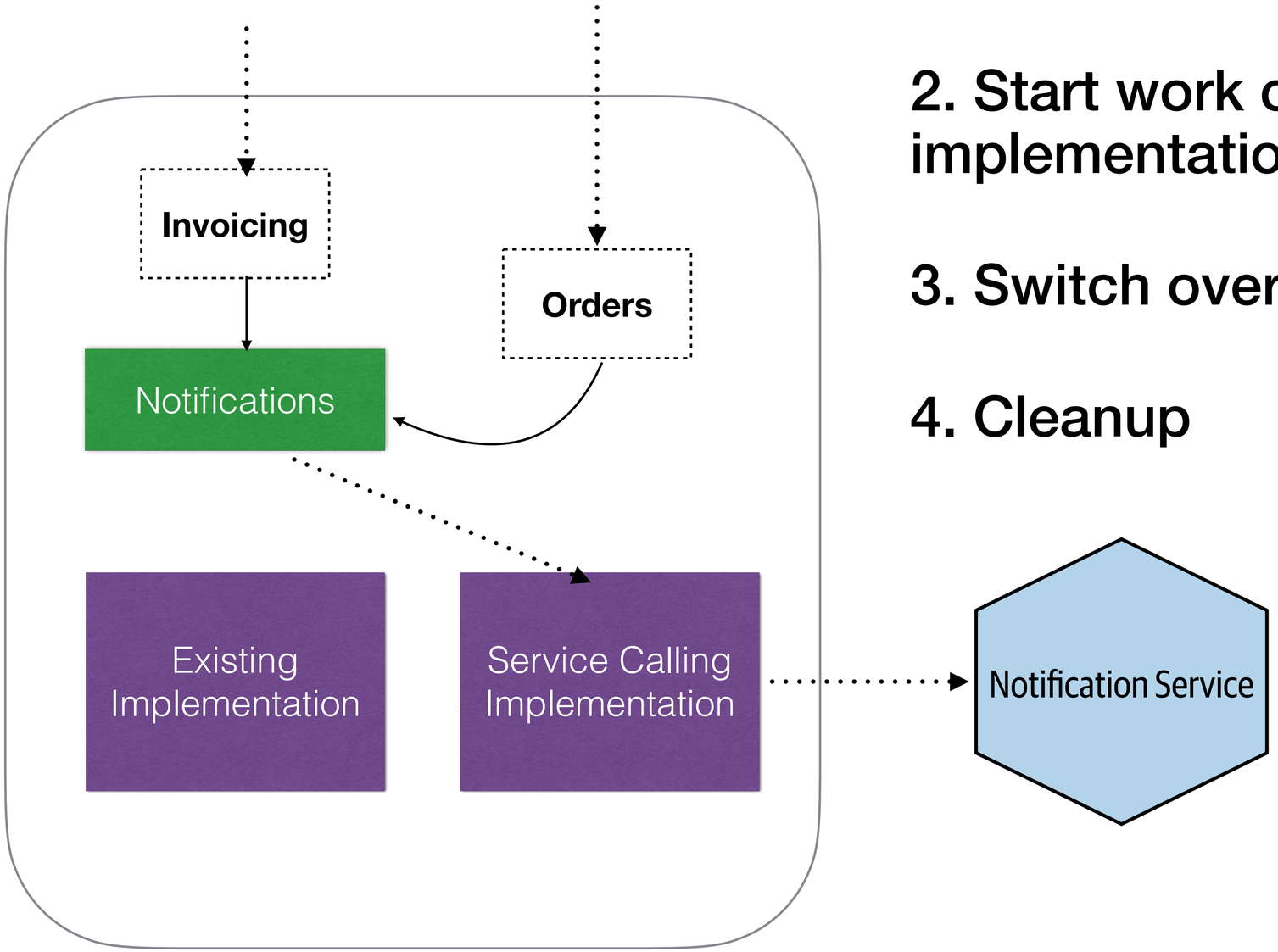



1. Create abstraction point

2. Start work on new service implementation

3. Switch over

# EXAMPLE



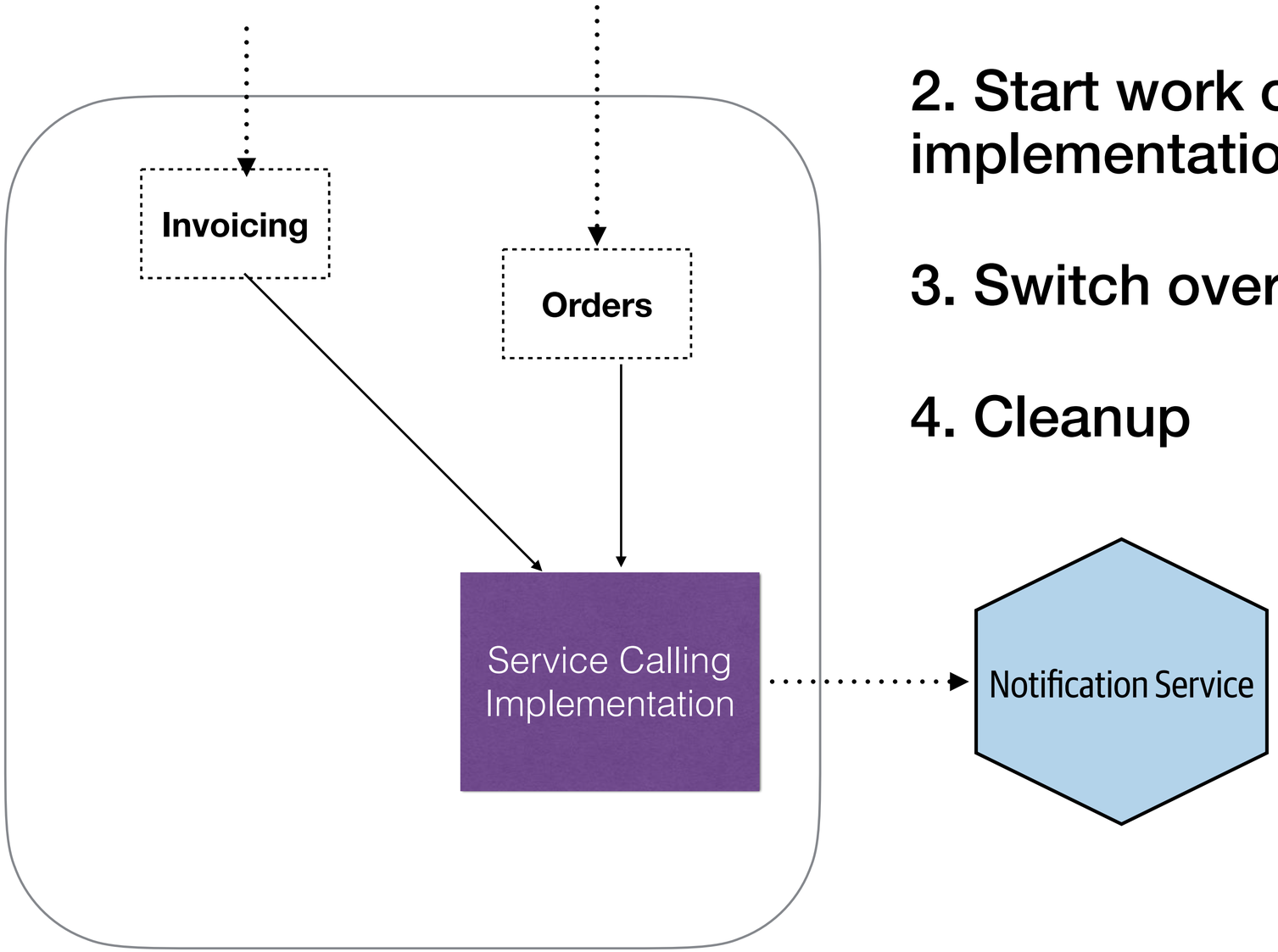

1. Create abstraction point

2. Start work on new service implementation

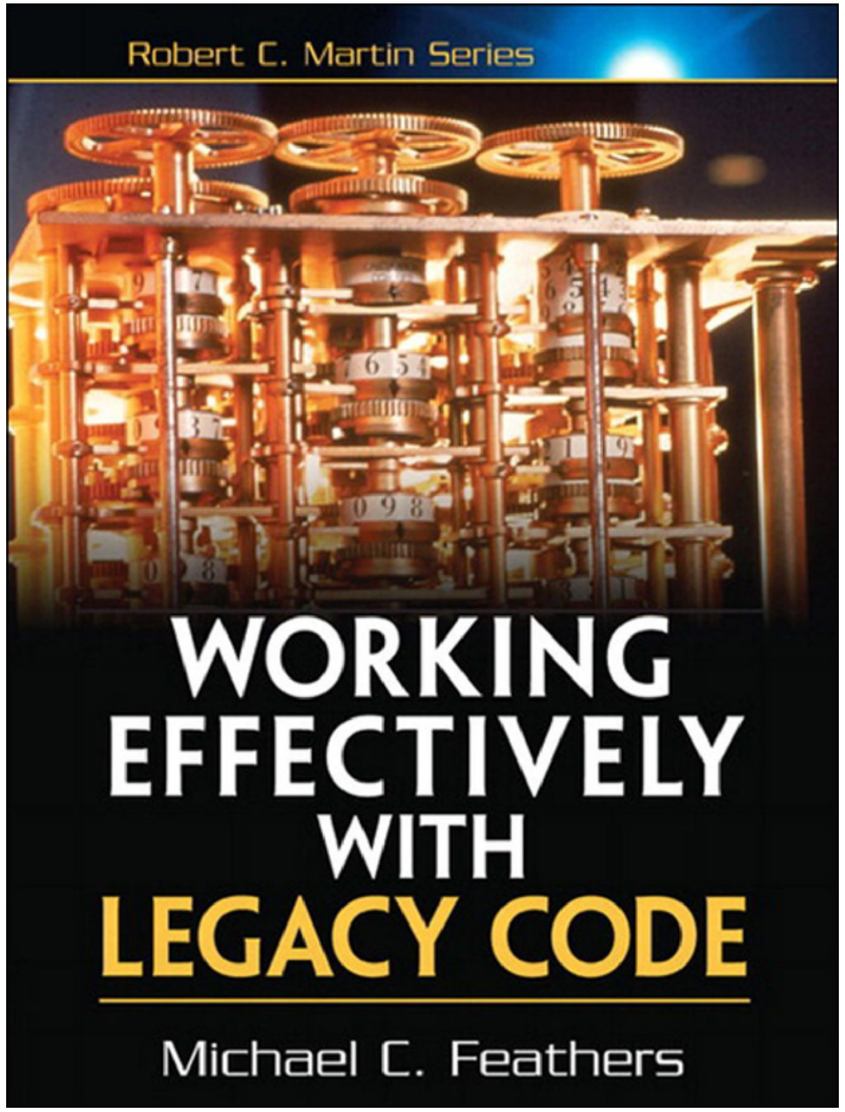
3. Switch over

4. Cleanup

# EXAMPLE




1. Create abstraction point
2. Start work on new service implementation
3. Switch over
4. Cleanup


Robert C. Martin Series
WORKING EFFECTIVELY WITH LEGACY CODE
Michael C. Feathers

# PATTERN: FEATURE TOGGLES


https://www.flickr.com/photos/paulcross/4333070249/

# PATTERN: FEATURE TOGGLES

## Feature Toggles (aka Feature Flags)

*Feature Toggles (often also refered to as Feature Flags) are a powerful technique, allowing teams to modify system behavior without changing code. They fall into various usage categories, and it's important to take that categorization into account when implementing and managing toggles. Toggles introduce complexity. We can keep that complexity in check by using smart toggle implementation practices and appropriate tools to manage our toggle configuration, but we should also aim to constrain the number of toggles in our system.*

09 October 2017

Pete Hodgson

Pete Hodgson is an independent software delivery consultant based in the San Francisco Bay Area. He specializes in helping startup

CONTENTS

expand



**Allow for functionality to be toggled off and on**

**https://martinfowler.com/articles/feature-toggles.html**

# PATTERN: FEATURE TOGGLES

## Feature Toggles (aka Feature Flags)

*Feature Toggles (often also refered to as Feature Flags) are a powerful technique, allowing teams to modify system behavior without changing code. They fall into various usage categories, and it's important to take that categorization into account when implementing and managing toggles. Toggles introduce complexity. We can keep that complexity in check by using smart toggle implementation practices and appropriate tools to manage our toggle configuration, but we should also aim to constrain the number of toggles in our system.*

09 October 2017

Pete Hodgson

Pete Hodgson is an independent software delivery consultant based in the San Francisco Bay Area. He specializes in helping startup

CONTENTS

expand



Allow for functionality to be toggled off and on

Can also be used to switch between alternative implementations

https://martinfowler.com/articles/feature-toggles.html

# PATTERN: FEATURE TOGGLES

## Feature Toggles (aka Feature Flags)

*Feature Toggles (often also refered to as Feature Flags) are a powerful technique, allowing teams to modify system behavior without changing code. They fall into various usage categories, and it's important to take that categorization into account when implementing and managing toggles. Toggles introduce complexity. We can keep that complexity in check by using smart toggle implementation practices and appropriate tools to manage our toggle configuration, but we should also aim to constrain the number of toggles in our system.*

09 October 2017

Pete Hodgson

Pete Hodgson is an independent software delivery consultant based in the San Francisco Bay Area. He specializes in helping startup

CONTENTS

expand



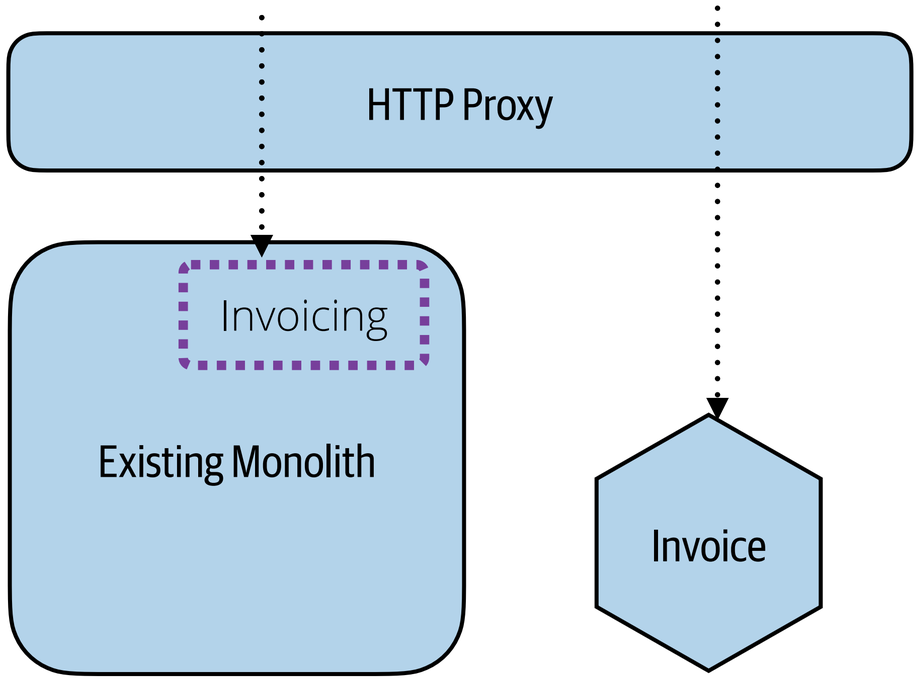
Allow for functionality to be toggled off and on

Can also be used to switch between alternative implementations

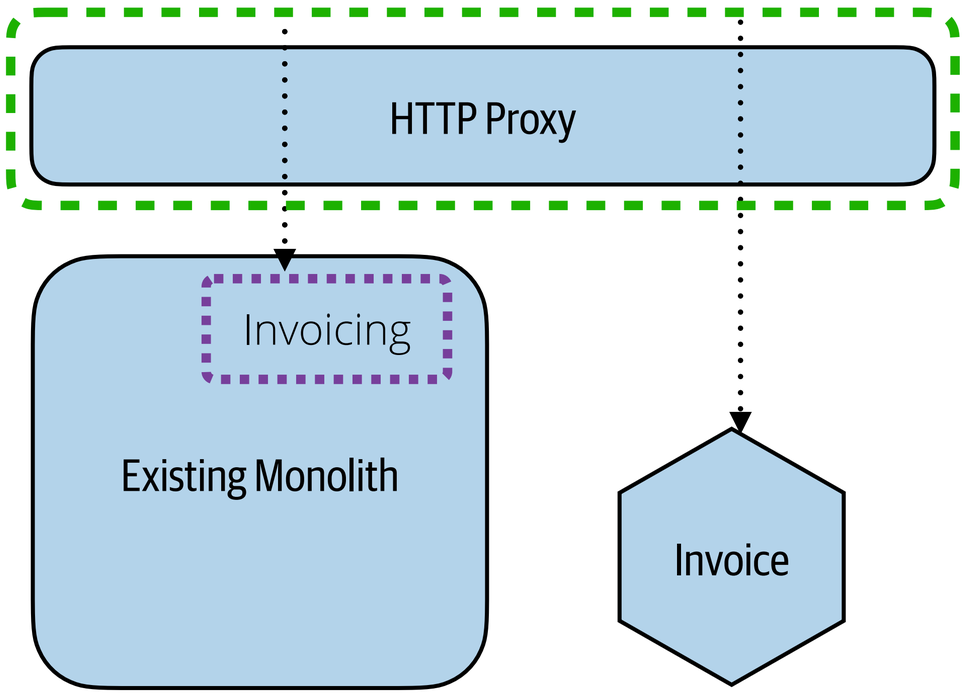
Changing toggled values doesn't require a rebuild or redeploy

https://martinfowler.com/articles/feature-toggles.html

# WITH THE STRANGLER FIG PATTERN

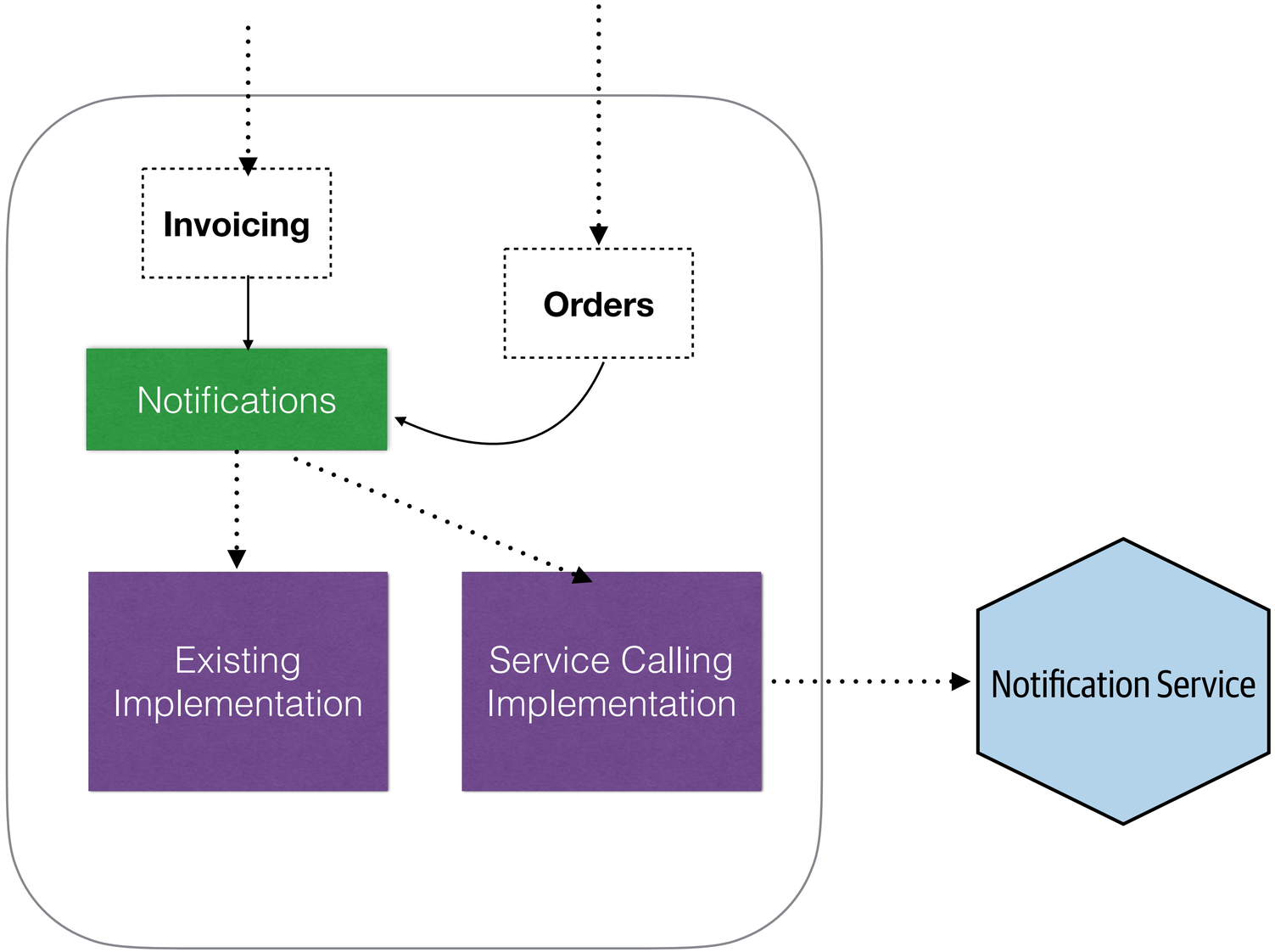


# WITH THE STRANGLER FIG PATTERN



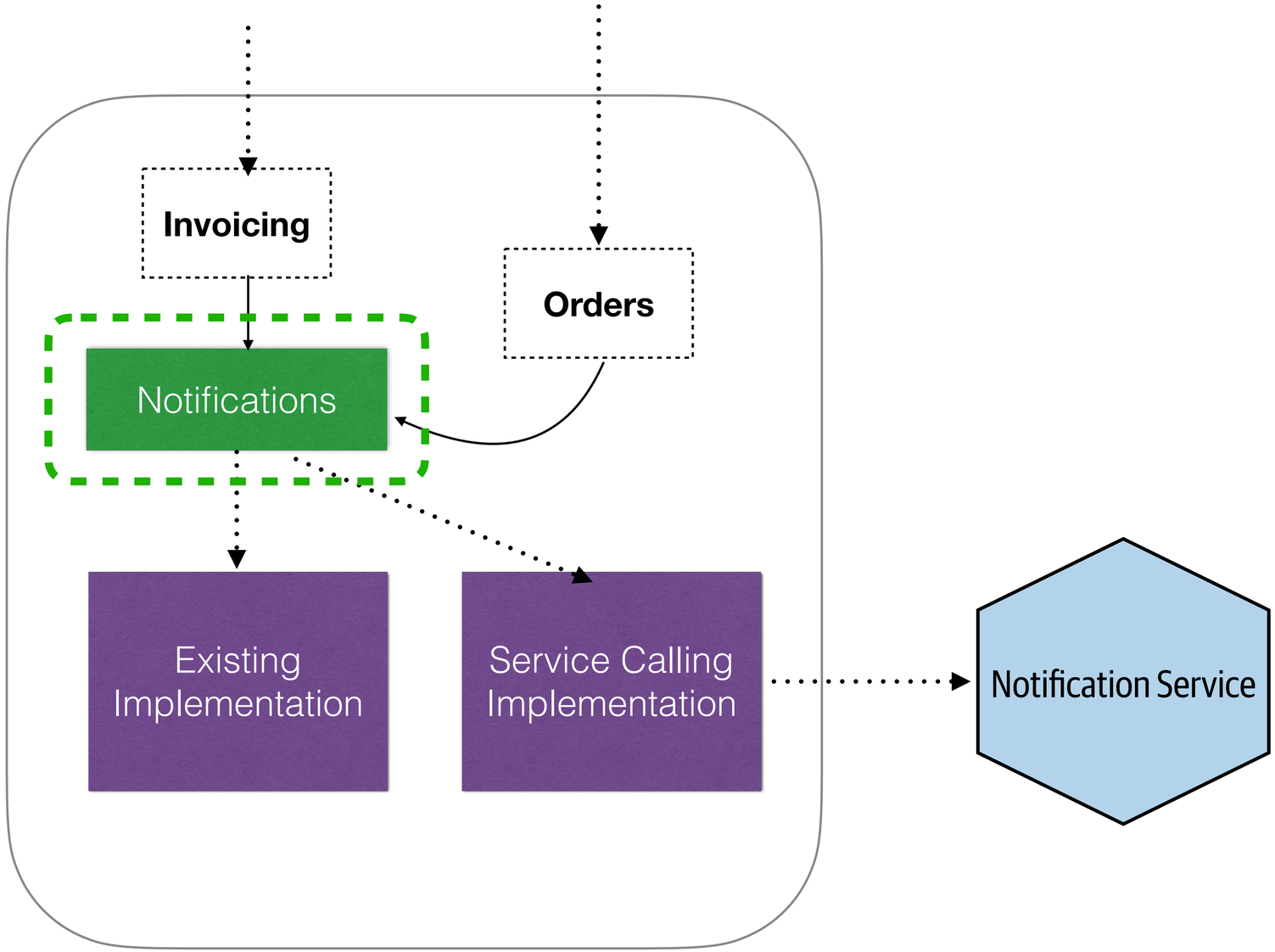

Implement a simple config-based toggle in the HTTP Proxy configuration

# WITH BRANCH BY ABSTRACTION



# WITH BRANCH BY ABSTRACTION

Use the abstraction to divert calls based on toggle value



# POLL: HAVE YOU USED THESE PATTERNS? (CAN SELECT MULTIPLE OPTIONS)

Strangler Fig

Branch By Abstraction

Feature Toggles

Parallel Run

## HOMEWORK

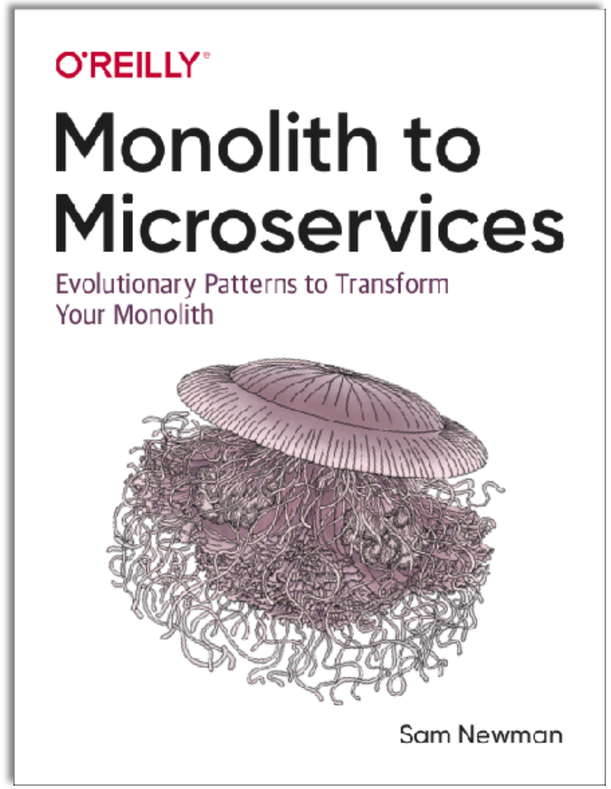
# Work out places where you can separate deployment from release

## HOMEWORK

Work out places where you can separate deployment from release

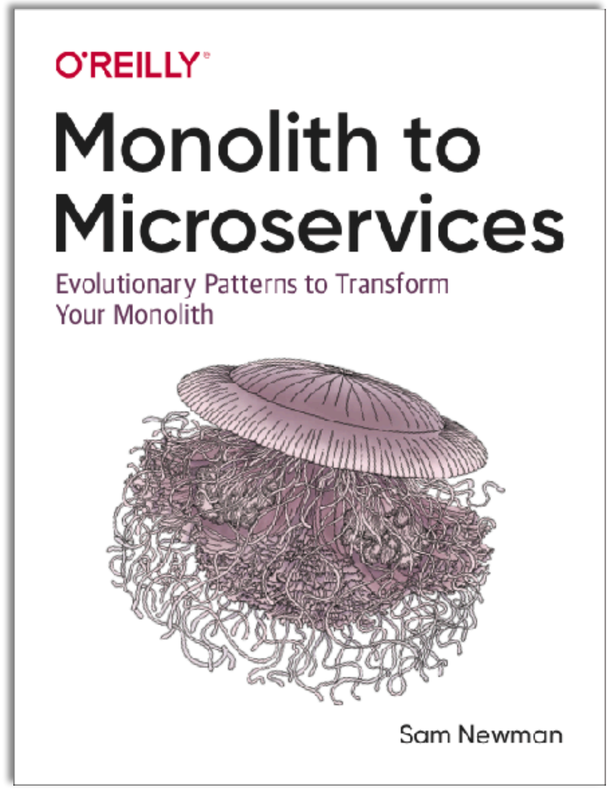
Can you make use of the Branch By Abstraction or Strangler Fig patterns?

# FURTHER READING

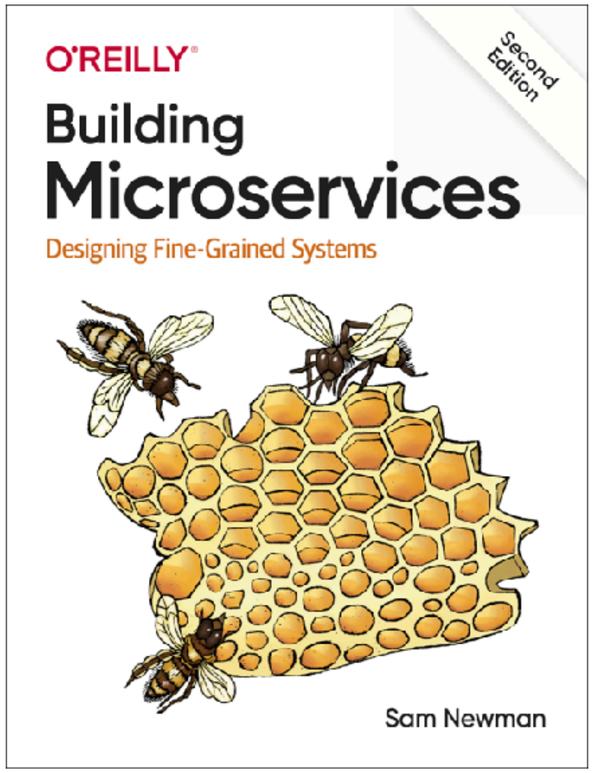



Chapters 2 & 3 as a recap of this class

# FURTHER READING

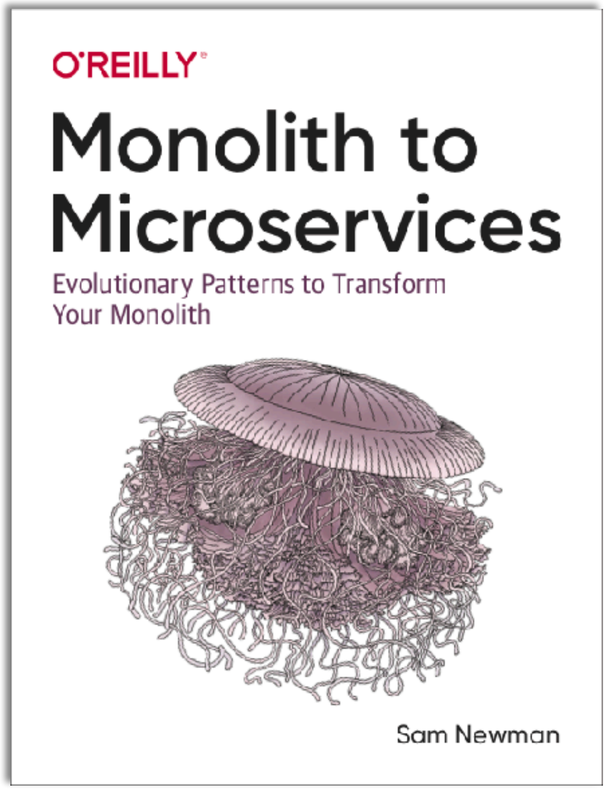



Chapters 2 & 3 as a recap of this class

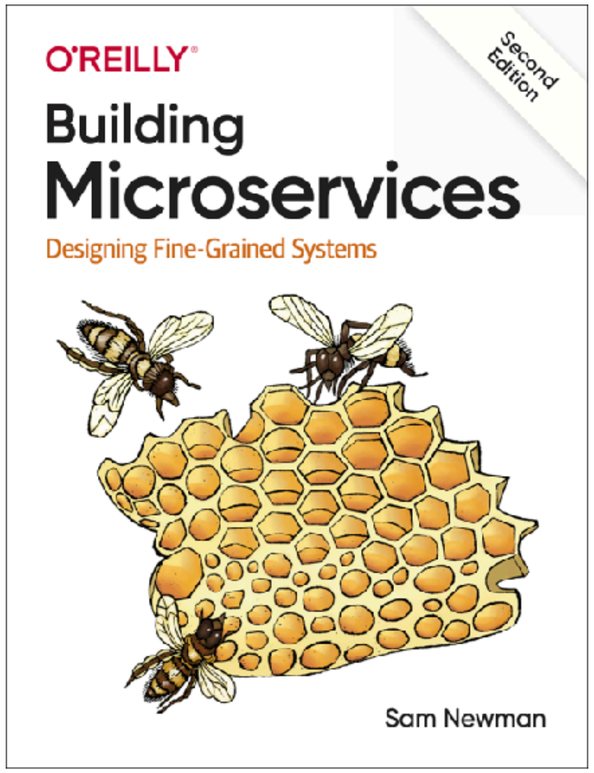



Chapter 14 (UI)

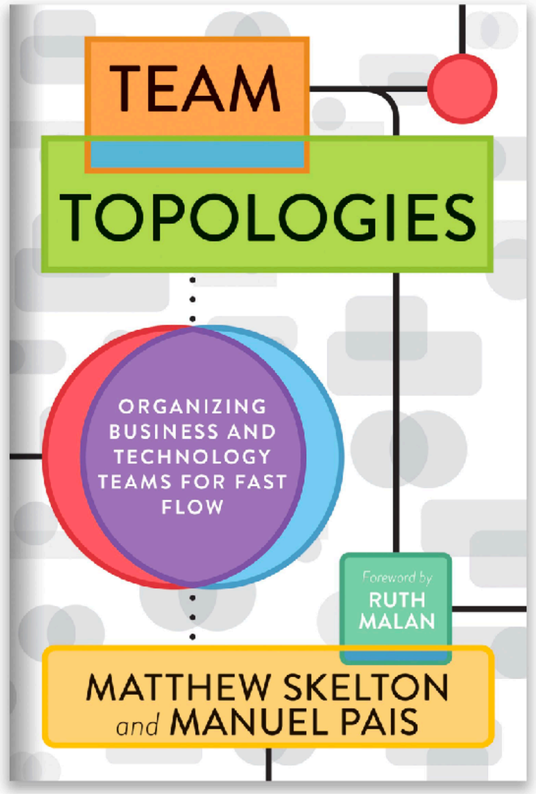
# FURTHER READING




Chapters 2 & 3 as a recap of this class




Chapter 14 (UI)



## NEXT WEEK…

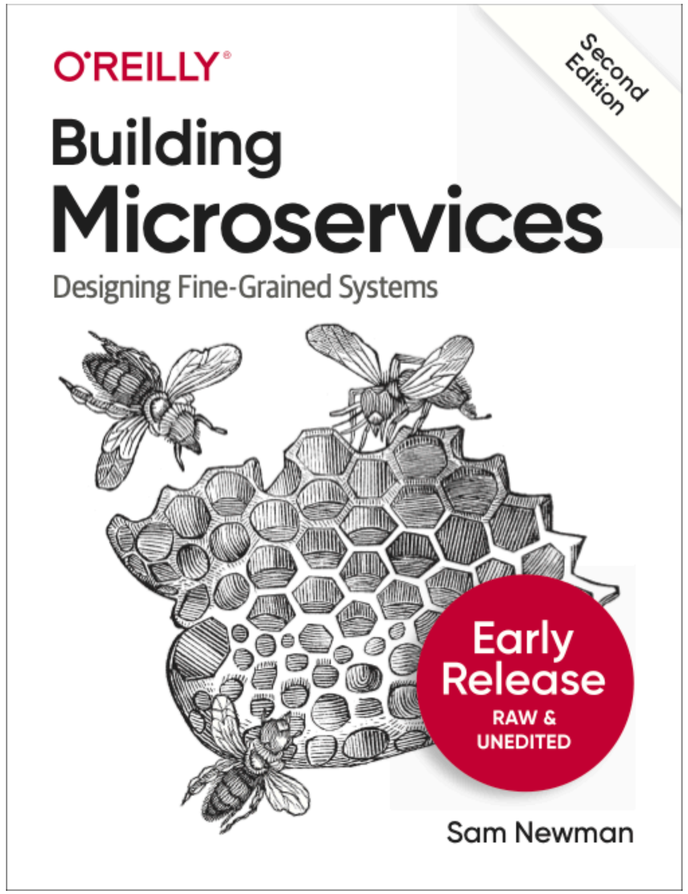
# Data and DB decomposition

## NEXT WEEK…

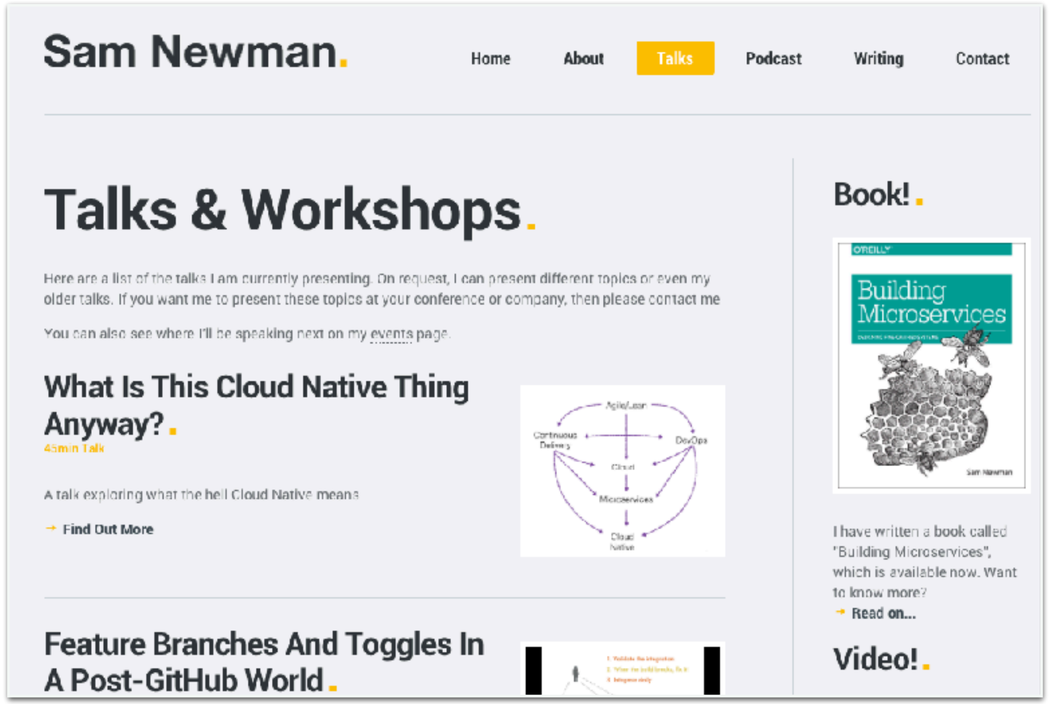
# Data and DB decomposition

# What else…?

# SEE YOU NEXT WEEK!




Sam Newman & Associates




@samnewman

https://samnewman.io/